(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 574 140 A1**

(12) EUROPEAN PATENT APPLICATION

(21) Application number : **93303743.4**

(22) Date of filing : **14.05.93**

(51) Int. Cl.[5] : **H04L 29/06**

(30) Priority : **29.05.92 US 891550**

(43) Date of publication of application : **15.12.93 Bulletin 93/50**

(84) Designated Contracting States : **DE FR GB**

(71) Applicant : **Hewlett-Packard Company**
3000 Hanover Street
Palo Alto, California 94304 (US)

(72) Inventor : **Thompson, Michael I.**
5921 Alan Drive
Rocklin, California 95677 (US)
Inventor : **Congdon, Paul T.**
4984 Creek View Way
Sacramento, California 95841 (US)
Inventor : **Burnett, John Lindsay**
20990 Valley Green Drive, Apt. No. 667
Cupertino, California 95014 (US)
Inventor : **Fiduccia, Frank**
Deceased (US)

(74) Representative : **Colgan, Stephen James et al**
CARPMAELS & RANSFORD 43 Bloomsbury Square
London WC1A 2RA (GB)

(54) Network adapter which places a network header and data in separate memory buffers.

(57) A computing system is connected to a network. The computing system (10) includes a main memory (11) and a network adapter (12). The network adapter (12) receives a network packet from the network. The network adapter (12) determines a location of a split between a header of the network packet and data contained in the network packet. The network adapter (12) places the header for the network packet into a first memory buffer in the main memory (11) and places the data contained by the network packet beginning at a second memory buffer in the main memory (11). A portion of the first memory buffer which is not filled by the header for the network packet is filled with pad data.

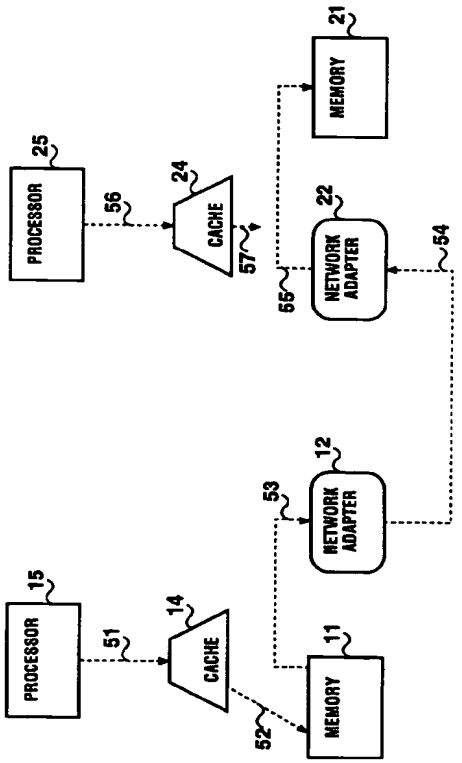

FIG. 2

EP 0 574 140 A1

BNSDOCID: <EP___0574140A1_I_>

Background

The present invention concerns a hardware system which performs insertion of network data checksums.

Most complex networks operate using several levels of protocol, each operating on a different layer of the network. For example, complex networks operating according to ISO OSI standard architecture include a physical layer, a link layer, a network layer and a transport layer. See, Proceedings of the IEEE, Volume 71, No. 12, December 1983.

The layers of protocol generally require various header fields to be included with data sent across a network. These header fields are variously used for such function as providing destination and source addresses, assuring efficient data flow, and detecting and correcting errors in data transmission. Typically, significant processor time is spent in generating the header fields, deciphering the information in the header fields and copying data.

Summary of the Invention

In accordance with the preferred embodiment of the present invention, a computing system is presented. The computing system is connected to a network. The computing system includes a main memory and a network adapter. The network adapter receives a network packet from the network. The network adapter determines a location of a split between a header of the network packet and data contained in the network packet. The network adapter places the header for the network packet into a first memory buffer in the main memory and places the data contained by the network packet beginning at a second memory buffer in the main memory. A portion of the first memory buffer which is not filled by the header for the network packet is filled with pad data.

In the preferred embodiment of the present invention, the network adapter is able to determine the split between the header and the data in a network packet by the use of a split offset count which the network adapter determines from the network packet. Using this split offset count, and the size of the first memory buffer, the network adapter can initiate a first count based on the split offset count, and a second count based on the size of the first memory buffer. For every byte of header sent from the network packet to the first memory buffer the network adapter decrements both the first count and the second count. When the first count reaches zero, the network adapter sends pad data to the first memory buffer and continues decrementing the second count for every byte of pad data sent from the network packet to the first memory buffer. The network adapter continues sending pad data to the first memory buffer until the second count reaches zero.

The preferred embodiment additionally includes, after a host processor builds a network packet header in the main memory, adding, by the host processor, pad bytes at the front of the network header and a count of the pad bytes. The resulting network packet, including the pad bytes and the count of the pad bytes, starts at a predetermined multi-byte boundary. A DMA transfer of the network packet is then performed from the main memory to the network adapter. The network adapter removes the count and the pad bytes from the network packet before transferring the network packet over the network.

The present invention simplifies the handling of the network packets by a system processor. By placing the data beginning at a new memory buffer, the system processor can forward the data to an application merely by forwarding to the application a pointer to the data. This eliminates the need of a time consuming copy of the data to the application memory space. Also, the addition of pad bytes and a count to an outgoing network packet can eliminate a copy of data in preparation to perform a DMA transfer from main memory to a network adapter.

Brief Description of the Drawings

Figure 1 shows a block diagram which shows two computer systems connected to a network.

Figure 2 shows data flow through a network in accordance with a preferred embodiment of the present invention.

Figure 3, Figure 4, Figure 5 and Figure 6 show sample headers for messages sent in accordance with a preferred embodiment of the present invention.

Figure 7 shows a block diagram of a network adapter in accordance with a preferred embodiment of the present invention.

Figure 8 shows a header of a network packet sent to a network adapter in accordance with a preferred embodiment of the present invention.

Figure 9 shows pad added to a packet constructed in memory in accordance with a preferred embodiment of the present invention.

Figure 10 is a simplified diagram showing a network adapter receiving a network packet from a host in accordance with a preferred embodiment of the present invention.

Figure 11 is a simplified diagram showing a checksum being added to a network packet in accordance with a preferred embodiment of the present invention.

Figure 12 is a simplified diagram showing a network packet being received by a network adapter from a network in accordance with a preferred embodiment of the present

Figure 13 shows a network packet stored in a main memory in accordance with a preferred embodiment of the present invention.

BNSDOCID: <EP___0574140A1_I_>

Figure 14 is a simplified diagram showing a network packet being sent by a network adapter to a host system in accordance with a preferred embodiment of the present invention.

Figure 15 shows a block diagram of a checksum logic cell array in accordance with the preferred embodiment of the present invention.

Description of the Preferred Embodiment

Figure 1 is a simplified block diagram which shows a computer system 10 connected to a computer system 20 over a network 30. Computer system 10 includes a processor 15, a cache 14, a memory 11 and a network adapter 12. A memory bus 13 connects processor 15 (through cache 14), memory 11 and network adapter 12. Network adapter 12 serves as an interface to network 30. Computer system 20 includes a processor 25, a cache 24, a memory 21 and a network adapter. A memory bus 23 connects processor 25 (through cache 24), memory 21 and network adapter 22. Network adapter 22 serves as an interface to network 30. The present invention concerns performance of the network data to simplify the assembly and deciphering of header fields for data which is sent across network 30.

Figure 2 shows data flow of a message which is transferred across network 30. A data path 51 and a data path 52 represent the flushing (or writing through) from cache 14 to memory 11 any information in cache 14 which will be used as part of the header or data forming the message. A data path 53 represents a DMA transfer operation from memory 11 to network adapter 12. A data path 54 represents data flow across network 30. A data path 55 represents a DMA transaction from network adapter 22 to memory 21. A data path 56 and a data path 57 represents an invalidation of memory locations within cache 24 which contain data made stale by the DMA transaction from network adapter 22 to memory 21.

In order to achieve the simplified data path for messages, network adapter 12 and network adapter 22 perform three operations normally performed by processor 15 and processor 25, respectively. The following discussion sets out how these operations are performed by network adapter 12; however, since network adapter 12 and network adapter 22 are identical in performance, the discussion equally applies to the operation of network adapter 22.

The first operation is the generation and insertion of network data checksums by network adapter 12. The implementation of the checksum calculation by network adapter allows the generation and insertion of the checksum with virtually no added ov rhead or tim incurred by processor 15.

In th outbound directi n, processor 15 provides network adapter 12 checksum control information which indicates the prop r method of ch cksumming and the location to insert the result. This control information is prepended to the packet in the DMA data stream traveling along data path 53. As the data is transferred from memory 11, network adapter 12 calculates the checksum. When the transfer is complete, network adapter 12 inserts the checksum into th proper location within the packet before transmitting the packet on network 30.

In the inbound direction, network adapter 12 decodes the packet header and programs the checksum control information directly into internal registers. The network adapter 12 calculates the checksum as it transfers the packet to memory 11. When the network adapter 12 completes the calculation of the checksum, network adapter 12 appends the result to the data stream that is being transferred to the memory 11. The processor 15 compares this checksum result against the packet checksum to verify th data.

The second operation which the network adapters performs is the automatic separation of headers and data during the transfer of incoming packets from network adapter 12 to memory 11. Splitting the header and data during the transfer allows the data to be placed on a page aligned boundary in host memory. Memory pages can be delivered to the proper application by virtual page remapping. This process is accomplished by simple virtual pointer manipulation and eliminates the need to copy the data once the packet has been transferred to memory 11 by network adapter 12.

Network adapter 12 determines the location of the header/data split and programs the DMA hardware with this value. The DMA hardware counts down the bytes of the header until the split location is reached. By adding pad data, the DMA hardware assures that the beginning of the data portion of th packet will fall on a memory page boundary. This process is run in parallel with the checksum process described above.

The third operation which network adapter 12 performs is the alignment of network headers. This is accomplished by the insertion of pad bytes based on specific values found in the network link header. Many processors are unable to access multi-byte fields which are not aligned to the corresponding multi-byte boundary. A processor would have to copy the bytes to an auxiliary buffer to access the data. Network adapter 12 eliminates the need for this copy.

Network adapter 12 searches the incoming byte stream for specific values in the destination SAP (DSAP) field of the network link header. The hardware will insert betwe n 0 and 3 pad bytes betw en the destination SAP and source SAP fields based on the value found in the destination SAP field. Subsequ nt headers will then be 4-byte aligned in the data stream.

Figures 3 through 6 show sample headers for

BNSDOCID: <EP___0574140A1_I_>

messages sent in accordance with a preferred embodim nt of the pres nt invention. Figure 3 shows an FDDI snap header 110 before th insertion of DSAP pad bytes. Media header 110 includes a one byte frame control (FC) field 111. A six-byte destination address field 112 gives the media address of a station receiving the packet. A six-byte source address field 113 gives the media address of a station sending the packet.

A one byte Destination Service Access Point (DSAP) field 114 defines the type of service being used and therefore the format of the media header. If DSAP equals AA_{hex} then media header 110 is a SNAP header and three bytes of pad will be inserted after DSAP field 114. If DSAP equals FC_{hex}, media header 110 is an HP expansion SAP and one pad byte will be inserted after DSAP field 114. In the present invention all other DSAP values result in the insertion of no pad bytes. This is because other headers, for example, an FDDI 802.2 header, do not require additional pads to be inserted.

A one byte SSAP field 116 contains the source service access point. LLC 802.2 Type 1 Control (CTRL) field 117, in the preferred embodiment, is always set to 3 which indicates unnumbered information. A three byte organization ID field 118 is not processed by the preferred embodiment of the present invention. A two byte SNAP type field 119 is not processed by the preferred embodiment of the present invention.

After the above-described fields of the link header, an IP header 120 is followed by a TCP header or a UDP header. IP header 120 is twenty or more bytes in length. A TCP header is twenty or more bytes in length. A UDP header is eight bytes in length.

In order to align the headers, the preferred embodiment of the present invention inserts between 0 and 3 pad bytes between DSAP field 114 and SSAP field 116, based on the value found in the DSAP field. Subsequent headers will then be 4-byte aligned in the data stream. In media header 110, DSAP equals AA_{hex} indicating media header 110 is a SNAP header. Therefore, three bytes of pad 115 are inserted after DSAP field 114, as shown in Figure 4.

Figure 5 shows an FDDI HP expansion header 130 before the insertion of a DSAP pad byte. Media header 130 includes a one byte frame control (FC) field 131. A six-byte destination address field 132 gives the media address of a station receiving the packet. A six-byte source address field 133 gives the media address of a station sending the packet.

A one byte Destination Service Access Point (DSAP) field 134 defines the type of service b ing used and th refore the format of th media header. DSAP equals FC_{hex}, indicating media header 130 is an HP expansion SAP and one pad byt will be inserted aft r DSAP field 134. A one byte SSAP field 136 c ntains the sourc service access point. LLC 802.2 Typ 1 Control (CTRL) field 137, in the preferred embodiment, is always set to 3 which indicates unnumbered information. A three byte HP xpansion service access point (XSAP) spacing field 138 is reserved. A two-byte destination expansion service access point (DXSAP) field 139 and a two-byte source expansion service access point (SXSAP) field 140 are utilized as part of a particular protocol.

After the above-described fields of the link header, an IP header 141 is followed by a TCP header or a UDP header.

In order to align the headers, the preferred embodiment of the present invention inserts a pad byte between DSAP field 134 and SSAP field 136, based on the value found in the DSAP field. Subsequent headers will then be 4-byte aligned in the data stream. In media header 130, DSAP equals FC_{hex} indicating media header 130 is an HP expansion SAP header. Therefore, a single pad byte 135 is inserted after DSAP field 134, as shown in Figure 6.

Figure 7 shows a block diagram of a portion of logic used to implement network adapter 12 in accordance with the preferred embodiment of the present invention. Network adapter 12 is connected to a backplane DMA controller 31 of computer system 10 through a backplane bus 33. Backplane DMA controller 31 performs DMA between memory 11 and network adapter 12.

Network adapter 12 is connected to network 30 through a front plane controller 32. For example, in the preferred embodiment, network 30 is an FDDI network and frontplane controller 32 is a LAN controller such as a LAN Controller DP83261, available from National Semiconductor Corporation, a California corporation having a place of business at 2900 Semiconductor Drive, Santa Clara, California 95051.

A frontplane logic cell array (LCA) 45 serves to receive data from and send data to network 30 via frontplane controller 32. LAN controller 32 provides transmission and reception of data packets to and from network 30.

For outbound transfers, frontplane LCA 45 unpacks the 32 bit words from a DMA bus 49 into 8 bit bytes for transmission by LAN controller 32. Frontplane LCA 45 also looks at the first byte of the output stream, which contains a count of how many FC pad bytes have been inserted by processor 15, and then strips off the FC pad bytes and sends the rest of th packet to LAN controller 32. Frontplane LCA 45 loads an outbound first-in-first-out memory (FIFO) with data for transmission. Frontplane LCA 45 then controls the handshake with LAN controller 32 for transmission of the pack t.

For inbound transfers, frontplane LCA 45 handshakes data into an inbound FIFO, while watching the status lines from LAN controller 32 for error conditions which would force a flush of the incoming packet. Frontplane LCA 45 takes th byte stream from LAN

BNSDOCID: <EP___0574140A1_I_>

controller 32 and packs it into a 32 bit word stream for DMA bus 49. Whil frontplane LCA 45 packs the data stream, it also k eps track of the l ngth of the packet being received and inserts this length as part of the packet status at the end of the packet. Frontplane LCA 45 also scans the input stream for the DSAP field to determine what type of padding will be required to force the remaining headers and data to be aligned. Upon finding the DSAP field, frontplane LCA 45 inserts 0-3 bytes of DSAP pad. When frontplane LCA 45 detects the end of the packet, by sensing the EDRCVD line, it will insert the status of the packet, as sensed from LAN controller 32, and the length of the packet. It will insert 0-3 bytes of pad to insure that the status and length are contained in a single 32 bit word. The length field will not include the number of bytes padded to align the status and length word, the SLLW, or the checksum result added by checksum LCA 42.

A backplane logic cell array (LCA) 41 serves to receive data from and send data to memory bus 13 through backplane DMA controller 31.

A DMA bus 49 is separated from a processor bus 38 by a latch 46. A central processing unit (CPU) 37, a random access memory (RAM) 36, a nonvolatile RAM (NOVRAM) 35 and a read only memory (ROM) 34 are connected to processor bus 38. For example processor bus 38 is 32 bits wide. CPU 37 is a 68020 processor with a processor clock speed of 25 megahertz (MHz) available from Motorola Communications and Electronics Inc., having a business address at 801 Ames Avenue, Milpitas, California. CPU 37 is used to provide selftest functionality, chipset control of the frontplane (e.g. for initialization), connection management of the FDDI link, and other various functions. The code for the CPU 37 is contained in ROM 34. At startup, the CPU 37 will copy the code to RAM 36 and then execute the code from there. This will allow CPU 37 to execute with zero wait state instruction cycles.

A checksum logic cell array (LCA) 42, a DMA control logic cell array (LCA) 43 and a slot memory 44 are connected to DMA bus 49.

DMA bus 49, together with checksum LCA 42, DMA control LCA 43, backplane LCA 41 and slot memory 44 function as a data pipe to move data between LAN controller 32 and backplane DMA controller 31 with high throughput and low latency. The data pipe also provides the checksum hardware assist and manipulates the data to correct for improper alignment of headers, data, and buffers. A side feature is that the data pipe provides restricted access to the data stream by CPU 37 and does so with litt ffect on transfer p rformance.

Slot memory 44 is a block of fast static RAM that is designed to provide a bandwidth of 50 Mbytes/sec. This bandwidth is shared between LAN controll r 32, backplane DMA controller 31, and CPU 37 by time division multipl xing. Access to slot memory 44 is controlled ntirely by DMA controller LCA 43. Slot memory 44 is logically divided into 8K byt (enough for a maximum sized FDDI packet) slots into which packets are deposited. The slot concept provides a simpl method of memory management.

The main function of DMA controller LCA 43 is to manage slot memory 44. DNA controller LCA 43 accepts requests for data transfers with slot memory 44 and then generates the addresses and data strobes necessary to move data to the proper client. No other device has direct access to slot memory 44. This method of memory management guarantees all accesses are short and no device will holp up another.

DMA controller LCA 43 provides two DMA channels, one for transfers with backplane DMA controller 31 and one for transfers with LAN controller 32. DMA controller LCA 43 also acts as a proxy for access by CPU 37 to slot memory 44. DMA controller LCA 43 has a CPU address register, that CPU 37 can load, which is used when CPU 37 requests access to slot memory 44. When CPU 37 requests data, DMA controller LCA 43 fetches data from the location pointed to by the CPU address register and latches it into latch 46 for later access by CPU 37. DMA controller LCA 43 also has another address register that is used to allow checksum LCA 42 to insert a checksum into an outbound packet.

Checksum LCA 42 snoops data bus 49 during data transfers and calculates a checksum as data is being moved to/from backplane DMA controller 31 from/to slot memory 44. In order to perform the checksum operation, the various parameters of the checksum must first be programmed into the checksum LCA 42. This is accomplished by inserting th configuration into the data stream.

Checksum LCA 42 is configured with a Checksum Type (None, TCP, UDP), a Checksum Start Offset, a Checksum Stop Offset and a Checksum Insert Offset (used only for outbound data packets). All th checksum offset parameters are BYTE offsets of the packet.

In the preferred embodiment of the present invention, checksum LCA 42 handles only ARPA services. Checksum LCA 42 will correctly handle a start and stop offset that is any arbitrary byte offset. This can be done because of the simple nature of the ARPA checksums, but for other checksums (i.e. OSI) this may not be sufficient.

The Checksum Stop Offset value must be the exact offset where checksumming must stop. If the checksum is to run to the end of the packet, it still must hav the exact offset. Checksum LCA 42 will stop ch cksumming if the end of packet (EOP) bit is reached, but ch cksum LCA 42 will not know if all the bytes of the word are valid and it will assume that they are. Therefore, some garbage bytes may be included in the checksum if the stop ffset is not exact.

BNSDOCID: <EP___0574140A1_I_>

Figure 15 shows a block diagram f checksum LCA 42. A typ and status register 186 stores the checksum type. A start offset register 187 stores the start offset. A stop offset register 188 stores the stop offset. An insert offset register 189 stores the insert offset. A checksum accumulator 182 calculates the checksum of the data stream on DMA bus 49. A checksum is stored in a checksum register 184. A multiplexor 183 selects the results of checksum accumulator 182 or a value on DMA bus 49 to be placed in checksum register 184. A latch 185 is used to latch checksum 184 onto DMA bus 49. A control 181 controls operation of checksum LCA 42. In the block diagram shown in Figure 15 shows only a single checksum accumulator, additional checksum accumulators may be added in order to accommodate different checksumming algorithms. These may be multiplexed to checksum register 184.

Backplane LCA 41 is used to handshake data with backplane DMA controller 31, pack and unpack data, and provide proper alignment of the data transferred through backplane DMA controller 31.

The following describes an outbound transfer with checksum insertion. In the preferred embodiment, network adapter 12 is always in a read pending state. This allows an inbound packet to get to processor 15 quickly. Because of this, processor 15 must first notify network adapter 12 that an outbound transfer is coming so that network adapter 12 can get set up to accept it. Also, since the outbound packet is to have a checksum inserted into the data stream, the packet must not be transmitted until the checksum has been inserted. This forces the outbound packet to be staged in slot memory 44 until the checksum process is complete.

The following describes what must be done in order to insert a checksum and send an outbound packet. The outbound packet must be built in memory 11. Processor 15 must prepend an appropriate checksum control header to the packet.

For example, Figure 8 shows an outbound packet 60 built in memory 11. Outbound packet 60 includes a checksum control header 61, a link level header 62, an IP header 63, a transport header 64 and user data 65. Checksum control header is shown to include a start offset field 71, a stop offset field 75, an algo field 72, a direction field 73. an insert field 74 and an insert offset field 76. Start offset field 71 indicates the byte at which checksumming is to start. Stop offset field 75 indicates the stop offset, that is, the number of bytes which are to be checksummed. Algo field 72 indicates the checksum algorithm used (TCP, UDP, etc.). Directi n field 73 indicates the direction of data flow (inbound or outbound). Insert field 74 indicates whether the outbound packet is to have a checksum inserted. Ins rt offset field 76 indicates the location where a checksum is to be inserted.

Figure 9 shows how FC pad bytes 160 may be added in memory 11 to align the headers along multi-byte boundaries, for example, along 16 byte boundaries. In Figure 9, Link l vel head r 62 includes a one byte frame control (FC) field 161, six-byte destination address field 162, a six-byte source address field 163, a one byte Destination Service Access Point (DSAP) field 164, a one byte SSAP field 166, a Control (CTRL) field 167 and other fields (not shown). In order to allow alignment of the headers for DMA, FC pad bytes are added before FC field 161. An FC pad count field 159 indicates the number of FC pad bytes added.

The use of FC pads allows the networking protocol software operating on processor 15 to build its headers in main memory 11, without forcing the link header to fall on a particular multi-byte boundary. Generally, headers are built from back to front. The first byte of the header, therefore, is not necessarily aligned on any particular byte boundary. In the prior art, when a header was not properly aligned for DMA out of main memory 11, it would be necessary to copy the header before DMA to network adapter 12. In th preferred embodiment of the present invention, th first bytes of a DMA transfer are the FC count which signals the number of FC pad bytes which precede the header in order to allow the DMA to start at an aligned (i.e. cache line boundary) location. Additional FC pad bytes are added so that DMA transfer begins on a sixteen byte boundary.

Once the outbound packet is built in memory 11, backplane DMA controller 31 starts to move data from memory 11 to network adapter 12. DMA controller LCA 43 moves the data from backplane LCA 41 to slot memory 44, except for the checksum control 61, which contains configuration data for checksum LCA 42 and DMA controller LCA 43.

DMA controller LCA 43 must check insert field 74 of checksum control header 60 to determine if the packet is to have a checksum inserted or not. In the case where a checksum is to be inserted, outbound packet 60 packet must be stored in slot memory 44 until the checksum is inserted. This is illustrated by Figure 10.

While DMA controller LCA 43 moves the data from backplane LCA 41 to slot memory 44, checksum LCA 42 snoops the data on DMA bus 49, checksumming the data as it goes by. In the preferred embodiment, a sixteen bit add with carry is used. The checksumming starts at the byte designated by start offset field 71 and continues until either the stop offset is reached or the end of packet (EOP) bit is sensed. Checksum LCA 42 mask off bytes that are not part of th checksum. This all ws checksumming to start and stop n arbitrary byte boundari s. This works fin for ARPA services, but some modification to the algorithm will be needed for other types of checksum.

When the last word of data is latched by backplane LCA 41, as indicated by backplane DMA con-

BNSDOCID: <EP___0574140A1_I_>

troller 31, backplane LCA 41 so signals to checksum LCA 42. Upon receipt of the signal, checksum LCA 42 asserts a checksum value 77 on data bus 49 and signals DMA controller LCA 43 to write checksum value 77 into slot memory 44 at the offset given in insert offset field 76. This is illustrated by Figure 11. Also upon detecting the end of the packet, backplane DMA controller 31 interrupts processor 15. If processor 15 is done with outbound transfers, processor 15 posts a read to network adapter 12. If processor 15 wishes to do another outbound transfer, processor 15 proceeds with the next outbound transfer.

Once checksum value 77 is written into slot memory 44, DMA controller LCA 43 starts to move data from slot memory 44 to LAN controller 32 for transmission to network 30. Frontplane control LCA 45 unpacks the 32-bit data stream into an 8 bit data stream. This data stream contains a Frame Control byte and the preceding FC pad bytes. Frontplane control LCA 45 will strip off the FC pad bytes and send the remainder of the data stream to LAN controller 32.

An outbound transfer may also be sent without a checksum insertion. Since this outbound packet won't have to insert a checksum, the packet can be immediately streamed to LAN controller 32 for transmission. The lack of an insertion of a checksum does not mean that a checksum cannot be calculated for the packet. For example, if an IP fragment train is being processed, the first N packets will have a running checksum calculated and only the N+1 packet will have the total checksum inserted into it. The first N packets can be immediately streamed to LAN controller 32.

The following process is done to receive an inbound packet. The inbound packet starts arriving in LAN controller 32. LAN controller 32 signals frontplane LCA 45 that data is arriving. Frontplane control LCA 45 clocks the data into a buffer within frontplane control LCA 45. When the number of bytes received exceeds a fixed threshold, a signal is given to DMA controller LCA 43 to start moving data into slot memory 44.

Figure 12 illustrates an inbound packet 80 being moved into slot memory 44. As data is moving into slot memory 44, frontplane LCA 45 is looking for the DSAP field within the inbound packet. Once found, the frontplane LCA 45 inserts pad bytes, based on the value of the DSAP, to align the data and header portion of the packet.

After the pad bytes are inserted, DMA controller LCA 43 moves the remainder of the data to slot memory 44. When the end of the packet is reached, frontplane LCA 45 will append th status bits from LAN controller 32 and the l ngth of th packet to the nd of the data stream. Frontplane LCA 45 will also force the status/length word to be long word aligned by appending leading pads. It will ass rt the EOP bit on the last byte f the status/length word.

DMA controller LCA 43 moves the data into a slot in slot memory 44. Wh n th number of bytes moved exceeds a fixed threshold, for example 32 bytes, an interrupt is given to CPU 37 to signal that it can start reading the header from slot memory 44. CPU 37 will analyze the header to determine the checksum type, start and stop offsets. The CPU will also determin where to separate the header from data.

DMA controller LCA 43 has a counter which keeps track of the number of pending packets waiting to be analyzed by CPU 37.

CPU 37 analyses the header and writes the header/data split and the checksum information to slot memory 44 using the CPU address register within DMA controller LCA 43. If the checksum type is NONE, the start and stop fields are ignored and do not need to be written.

The first 3 words of the transfer are not given to backplane LCA 41, but are latched by checksum LCA 42 to load the checksum configuration for the current inbound packet.

The next word is loaded by backplane LCA 41 into an internal split offset counter within backplane LCA 41, as well as passing it on as data to processor 15. This internal split offset count is based on the header buffer length and is used to determine when the header has been sent to processor 15 and when to start padding data bytes to fill out the first buffer on the data chain.

Backplane LCA 41 unpacks the 32 bit data stream from midplane bus 49 into a 16 bit data stream for transfer to backplane DMA controller 31. It will continue to do this until it senses the End-Of-Packet (EOP) bit, which ends the DMA transfer.

The DSAP pads help insure that the header/data split is on a four-byte boundary, so it will not be necessary to handle odd byte alignments. If CPU 37 determines that the header/data split is not on an even byte boundary, the inbound packet must be transferred to memory anyway, and the alignment must be corrected by processor 15. If backplane LCA 41 detects an odd split offset, it will set an error indication in a status register accessible by processor 15.

As data is sent to memory 11, backplane LCA 41 decrements the split offset count and the header buffer length count. The header buffer length count is initialized to the number of bytes in a memory buffer. When the split offset count reaches zero, backplane LCA 41 will start sending pad data and decrementing a header buffer length count until it reaches zero. This serves to fill up the first buffer which is designed to contain only the header. At this time, backplane LCA 41 resumes sending data from slot m mory 44. This will be the data payload which will end up being page aligned. In an alt rnate preferred embodiment, when the split offset count reaches zero, backplan LCA 41 will start sending data to a next mem ry buffer without sending pad data. This alternate embodiment al-

BNSDOCID: <EP___0574140A1_I_>

lows more efficient transfers, and is preferred when the backplane DMA controller is able to implement it.

Figure 13 illustrates the resultant alignment in memory 11. In memory 11, a header buffer 91, a data buffer 92 and a data buffer 93 are shown. Header buffers are a small portion of memory. Data buffers 92 and 93 each represent a single page of memory 11. For example, each page of memory includes 2048 bytes of data. In order to assure that data is page aligned, pad data 102 is added after a packet header 101 to fill up header buffer 91. This assures that packet data 103 is placed beginning in data buffer 92. Additional packet data 104 may be placed in a following data buffer 93, if necessary.

When the last word of the data is latched into backplane LCA 41, the EOP bit will be set. This word is the Status Length Long Word (SLLW). Backplane LCA 41 now looks to see if this last 32 bits of data will be on an 8 byte boundary in memory 11. If not, it will send pad bytes to force this alignment. This assists processor 15 in finding the status and checksum information quickly. The SLLW will then be sent.

After backplane LCA 41 sends the SLLW, backplane LCA 41 will read in one more word from the pipeline data bus, which contains a checksum result 87. This is represented by Figure 14. Checksum result 87 is transferred to backplane DMA controller 31 with a signal that the DMA transfer is complete. This terminates the DMA transfer and backplane DMA controller 31 will generate an interrupt to processor 15 signaling the DMA transfer completion. Processor 15, upon interrupt, reads a status register within backplane LCA 41, which indicates the length (in words) of the inbound transfer. The status register will also indicate if any errors occurred on network adapter 12 and whether buffers exist on network adapter 12 to do more inbound or outbound processing.

The foregoing discussion discloses and describes merely exemplary methods and embodiments of the present invention. As will be understood by those familiar with the art, the invention may be embodied in other specific forms without departing from the spirit or essential characteristics thereof. Accordingly, the disclosure of the present invention is intended to be illustrative, but not limiting, of the scope of the invention, which is set forth in the following claims.

Claims

1. In a computing system (10) connected to a network (30), th c mputing system (10) including a main memory (11) and a network adapter (12), a method for forwarding to th main memory (11) a network packet (101, 103, 104) received by the n twork adapter (12) from the network (30), th method comprising th steps of:

   (a) placing, by the network adapt r (12), a header (101) for th network packet (101, 103, 104) into a first memory buffer (91) in the main memory (11); and,

   (b) placing, by the network adapter (12), data contained by the network packet (101, 103, 104) beginning at a second memory buffer (92) in the main memory (11).

2. A method as in claim 1 wherein step (a) includes transferring, by the network adapter (12), pad data (102) into the first memory buffer (91) to fill a portion of the first memory buffer (91) which is not filled by the header (101) for the network packet (101, 103, 104).

3. A method as in claim 1 additionally comprising the following step which is performed before step (a):

   (c) determining, by the network adapter (12), a split offset count from the network packet (101, 103, 104), the split offset count indicating length of the header (101).

4. A method as in claim 3 wherein step (a) additionally includes the following substeps:

   (a.1) initiating a first count based on the split offset count, and initiating a second count based on the size of the first memory buffer (91);

   (a.2) for every byte of header (101) sent from the network packet (101, 103, 104) to the first memory buffer (91) decrementing both th first count and the second count; and,

   (a.3) when the first count reaches zero, sending pad data (102) to the first memory buffer (91) and decrementing the second count for every byte of pad data (102) sent from the network packet (101, 103, 104) to the first memory buffer (91), the sending of pad data (102) to the first memory buffer (91) continuing until the second count reaches zero.

5. A method as in claim 3 wherein step (a) additionally includes the following substeps:

   (a.1) initiating a first count based on the split offset count;

   (a.2) for every byte of header (101) sent from the network packet (101, 103, 104) to the first memory buffer (91) decrementing the first count; and,

   (a.3) when the first count reaches zero, proceeding to step (b).

6. In a computing system (10) connected to a n twork (30), th computing system (10) including a main memory (11) and a network adapter (12), a method for forwarding to the main memory (11)

BNSDOCID: <EP___0574140A1_I_>

a network packet (101, 103, 104) from the network (30), the method comprising the steps of:

(a) receiving, by th network adapter (12), the network packet (101, 103, 104) from the network (30);

(b) determining, by the network adapter (12), a location of a split between a header (101) of the network packet (101, 103, 104) and data contained in the network packet (101, 103, 104);

(c) placing, by the network adapter (12), the header (101) for the network packet (101, 103, 104) into a first memory buffer (91) in the main memory (11); and,

(d) placing, by the network adapter (12), the data contained by the network packet (101, 103, 104) beginning at a second memory buffer (92) in the main memory (11).

7. A method as in claim 6 wherein step (b) includes determining, by the network adapter (12), a split offset count from the network packet (101, 103, 104), the split offset count indicating length of the header (101).

8. A method as in claim 8 wherein step (c) includes the following substeps:

(c.1) initiating a first count based on the split offset count, and initiating a second count based on the size of the first memory buffer (91);

(c.2) for every byte of header (101) sent from the network packet (101, 103, 104) to the first memory buffer (91) decrementing both the first count and the second count; and,

(c.3) when the first count reaches zero, sending pad data (102) to the first memory buffer (91) and decrementing the second count for every byte of pad data (102) sent from the network packet (101, 103, 104) to the first memory buffer (91), the sending of pad data (102) to the first memory buffer (91) continuing until the second count reaches zero.

9. In a computing system (10) connected to a network (30), the computing system (10) including a processor (15) and a main memory (11), a network adapter (12) comprising:

receiving means (44, 45) for receiving a network packet (101, 103, 104) from the network (30);

determining means (37), coupled to the receiving means (44, 45), for det rmining a location of a split b tw en a h ader (101) of the network packet (101, 103, 104) and data contained in th network packet (101, 103, 104);

transfer m ans (31, 41), coupled to the determining means (37), for transferring th network packet (101, 103, 104) to the main memory (11), the transf r means including first means for placing the header (101) for the network packet (101, 103, 104) into a first memory buffer (91) in the main memory (11) and for placing the data contained by the network packet (101, 103, 104) beginning at a second memory buffer (92) in the main memory (11).

10. A method for transferring a network packet (60) to a network adapter (12) for transfer over a network (30), the method comprising the steps of:

(a) building, in a main memory (11) by a host processor (15), a network packet header (110);

(b) adding, by the host processor (15), pad bytes at the front of the network packet header (110) and a count of the pad bytes so that the resulting network packet (60), including the pad bytes and the count of the pad bytes, starts at a predetermined multi-byte boundary; and,

(c) performing a DMA transfer of the network packet (60) from the main memory (11) to the network adapter (12).

BNSDOCID: <EP___0574140A1_I_>

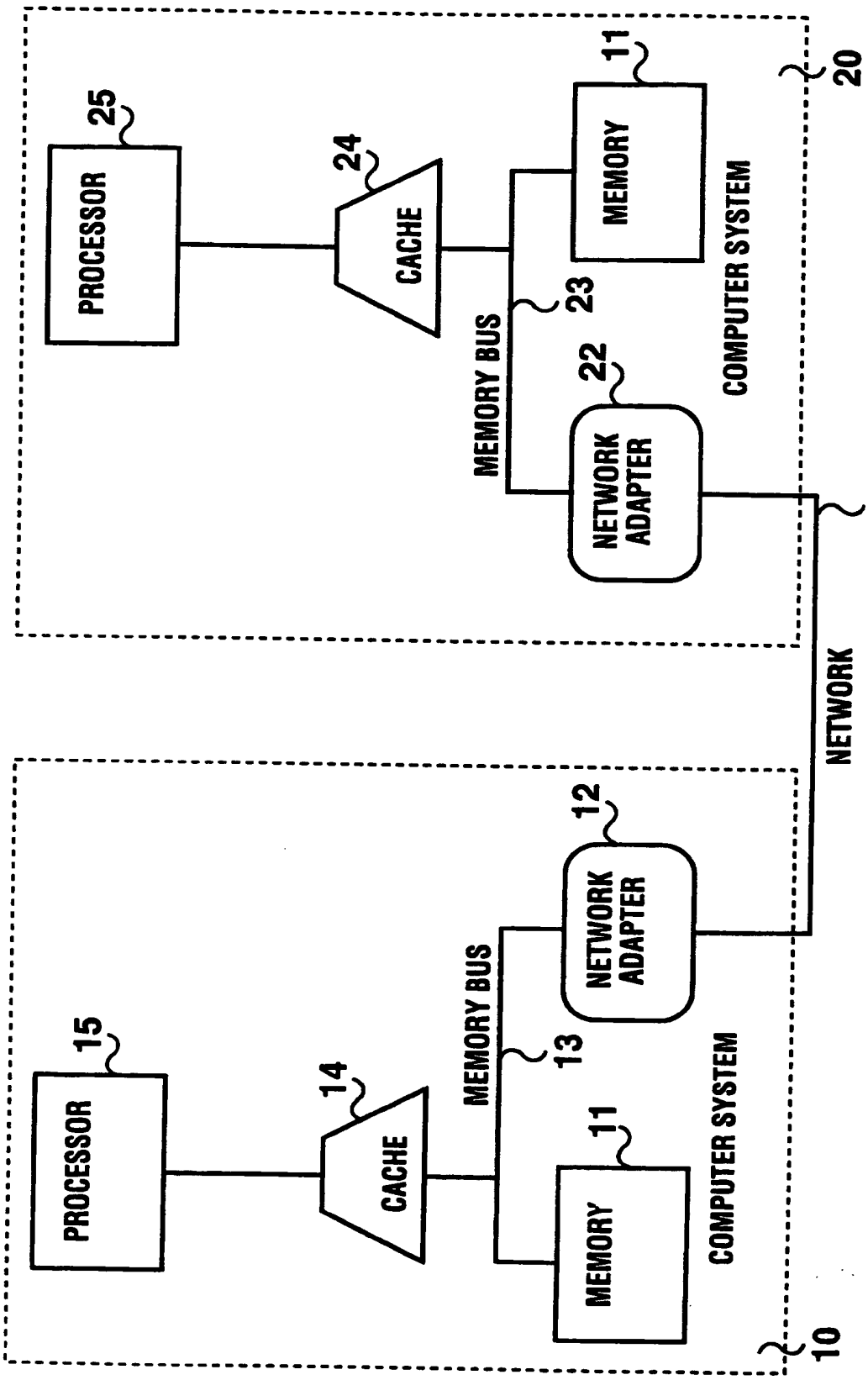

FIG. 1

BNSDOCID: <EP___0574140A1_I_>

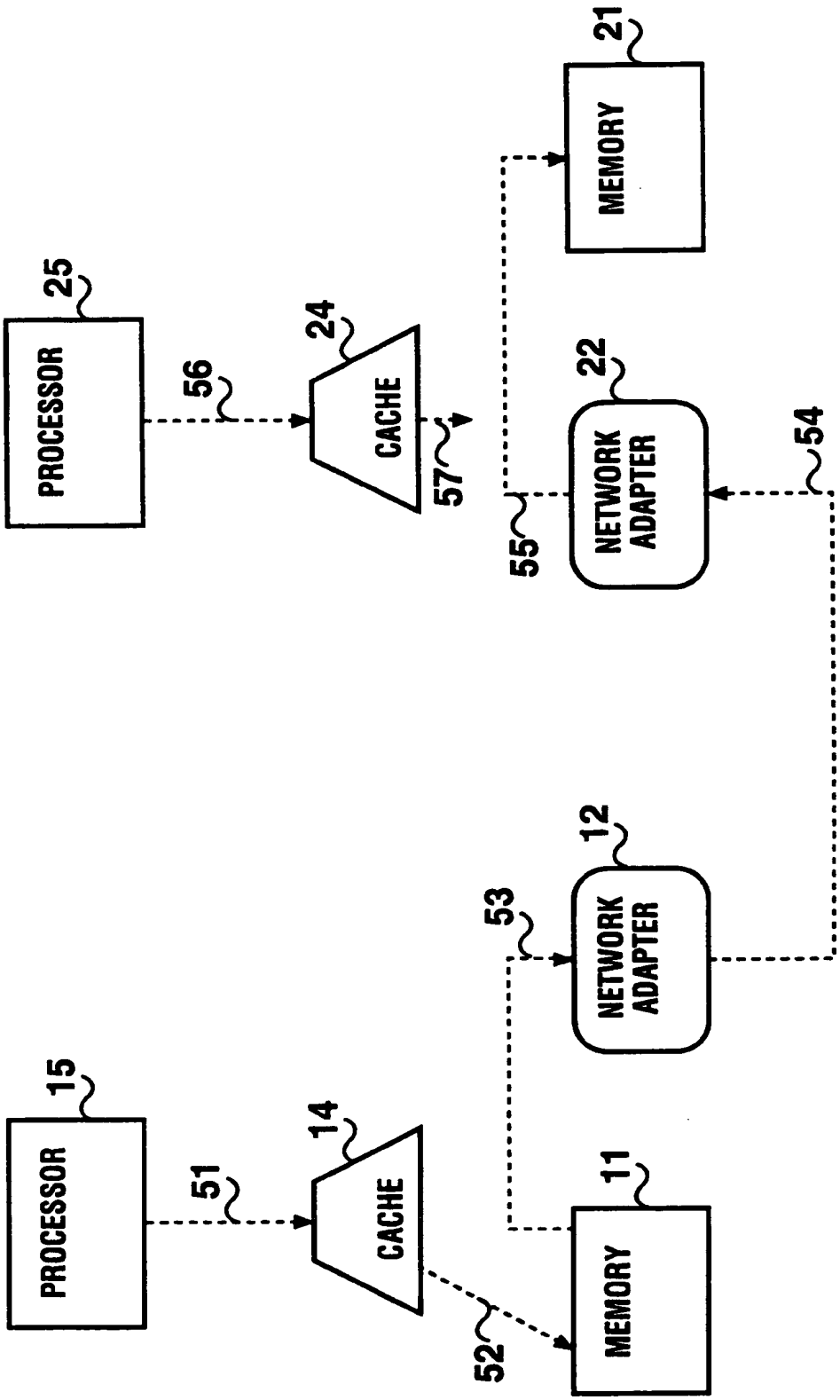

FIG. 2

BNSDOCID: <EP___0574140A1_I_>

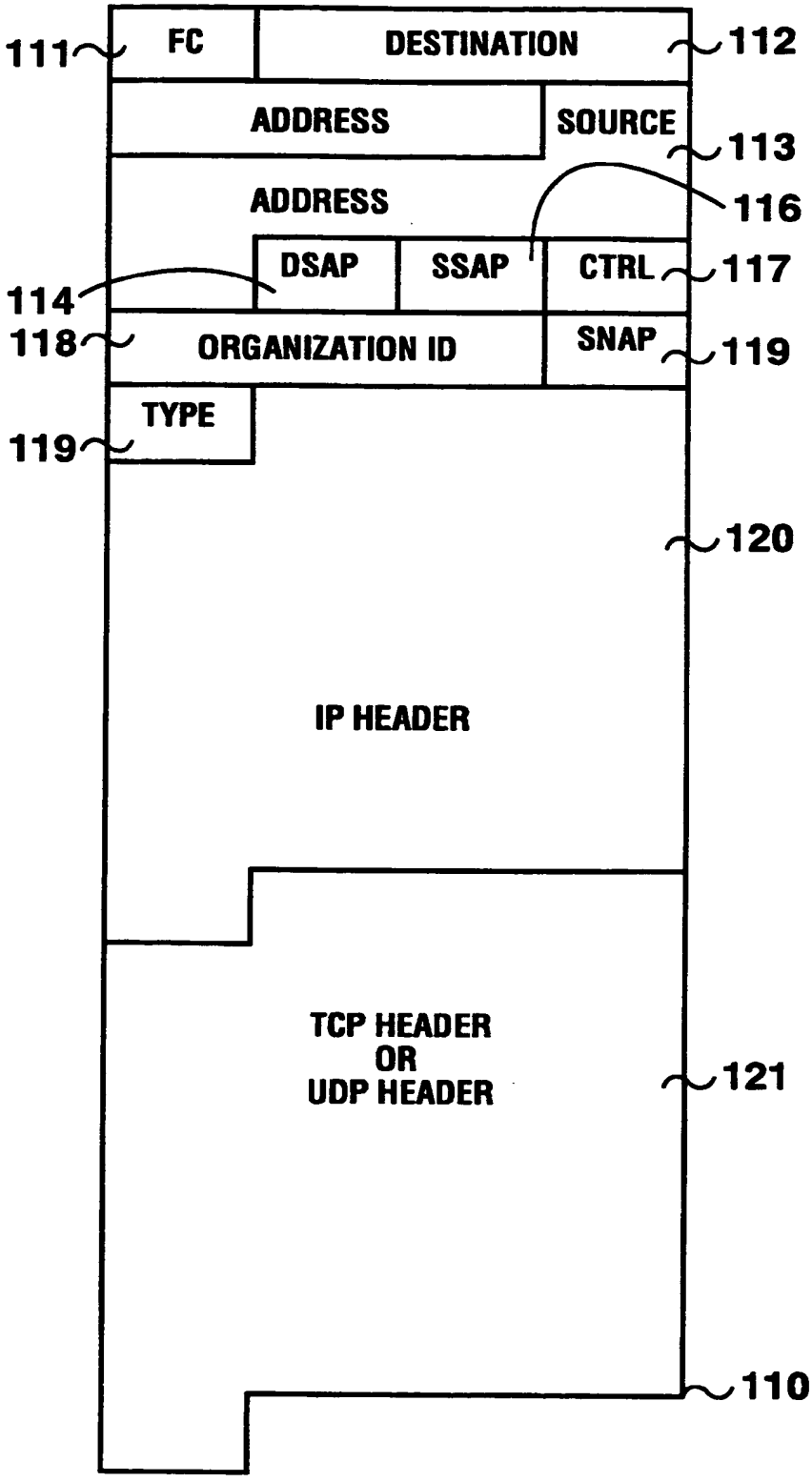

FIG. 3

BNSDOCID: <EP___0574140A1_I_>

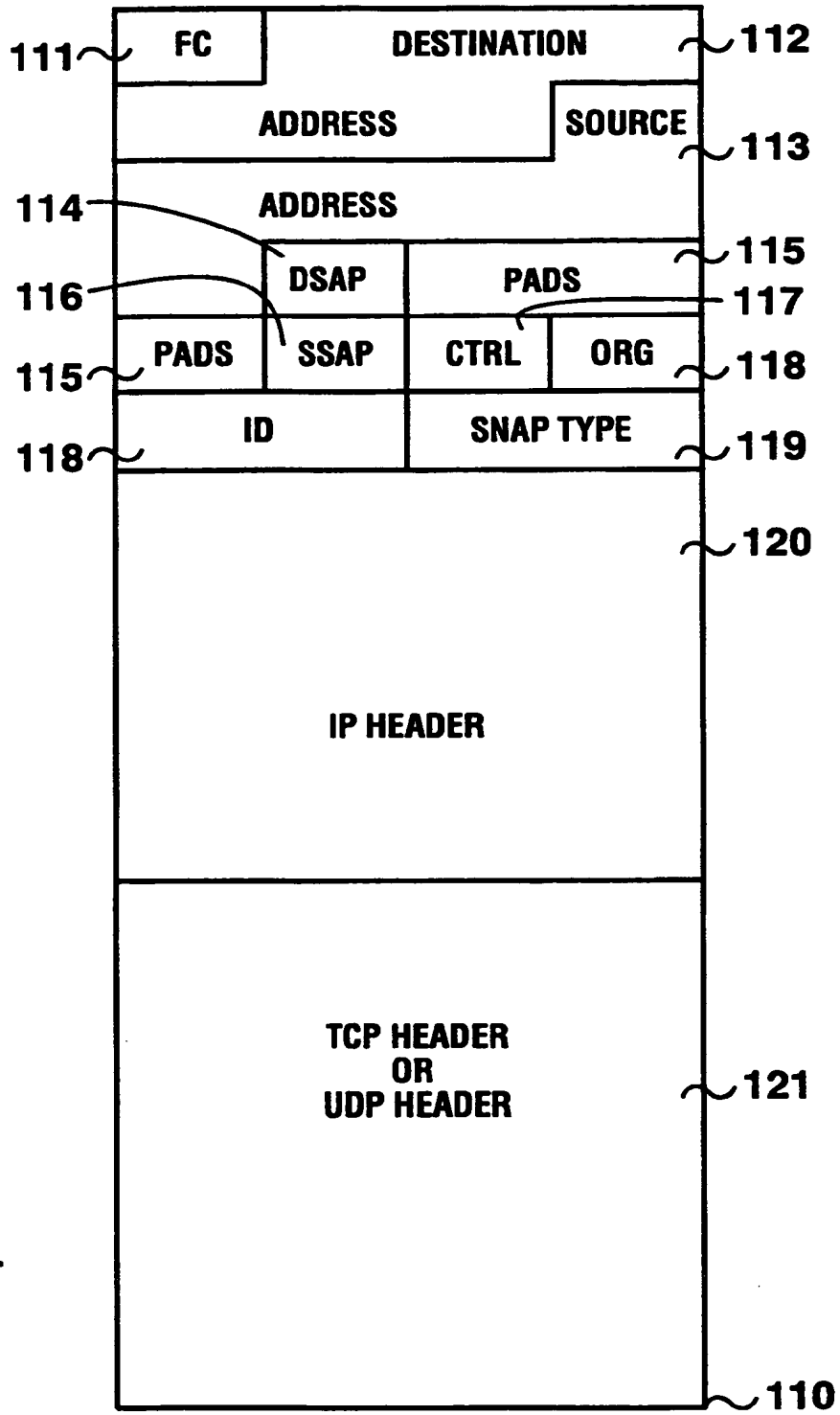

FIG. 4

BNSDOCID: <EP___0574140A1_I_>

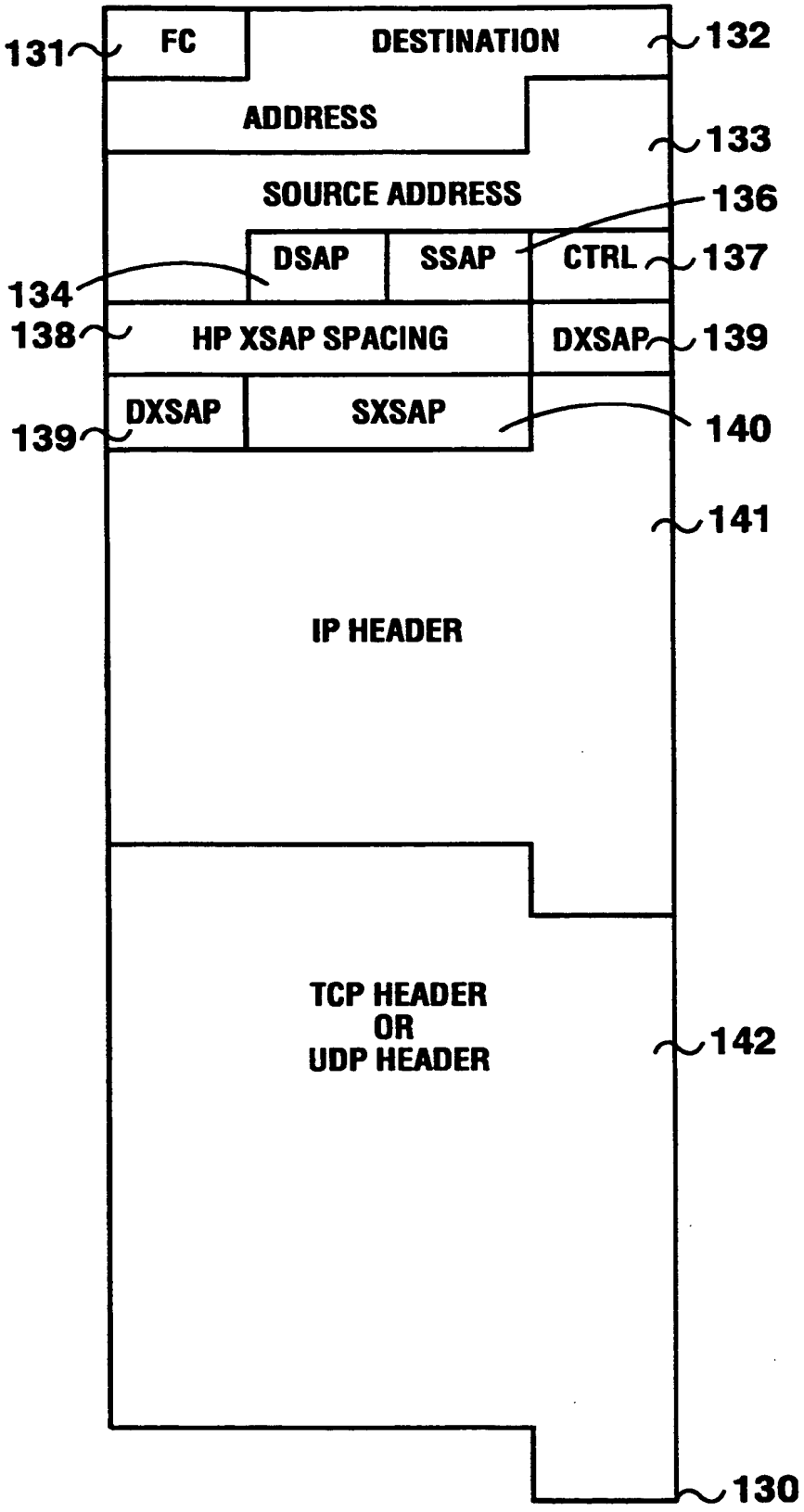

FIG. 5

BNSDOCID: <EP___0574140A1_I_>

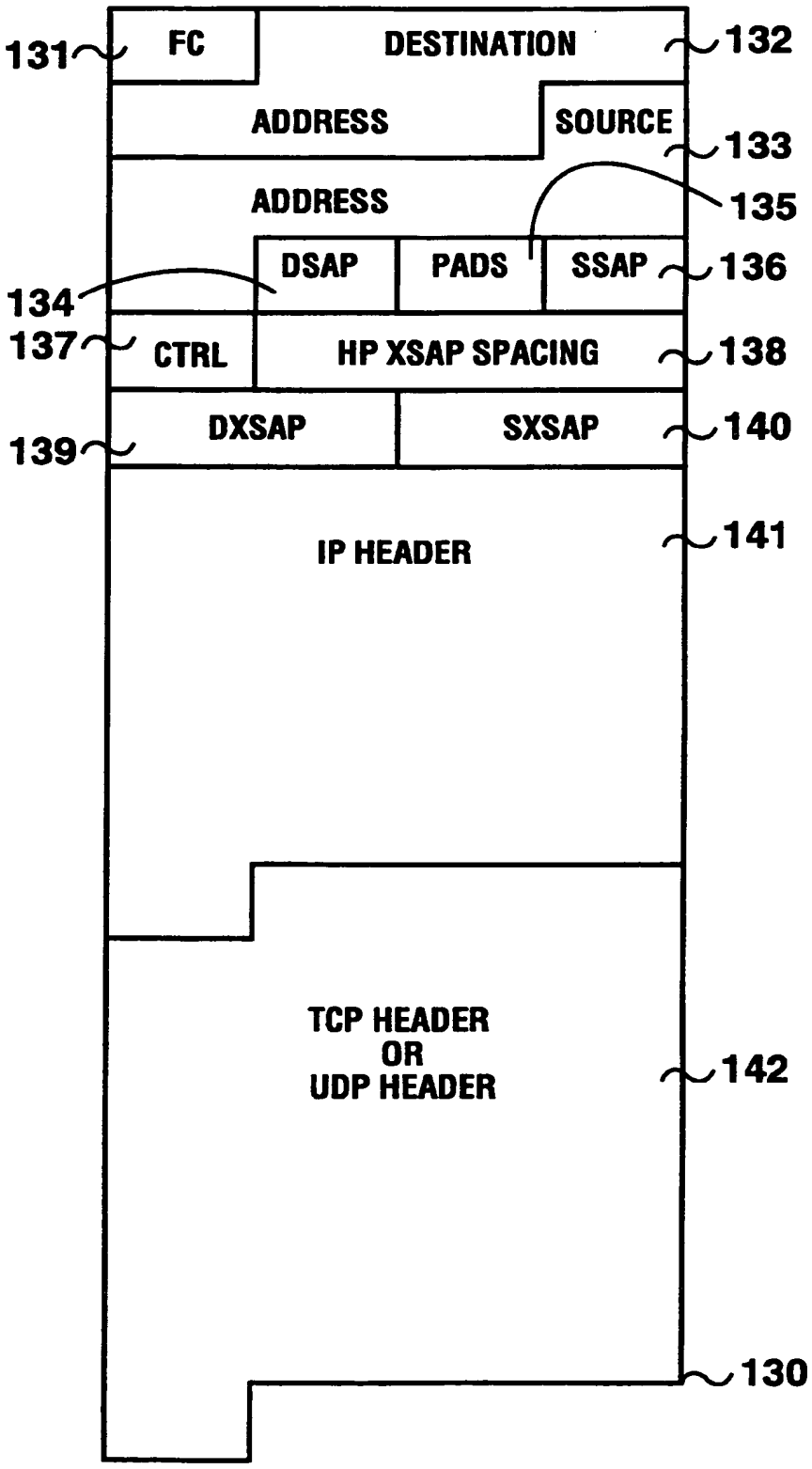

FIG. 6

BNSDOCID: <EP___0574140A1_I_>

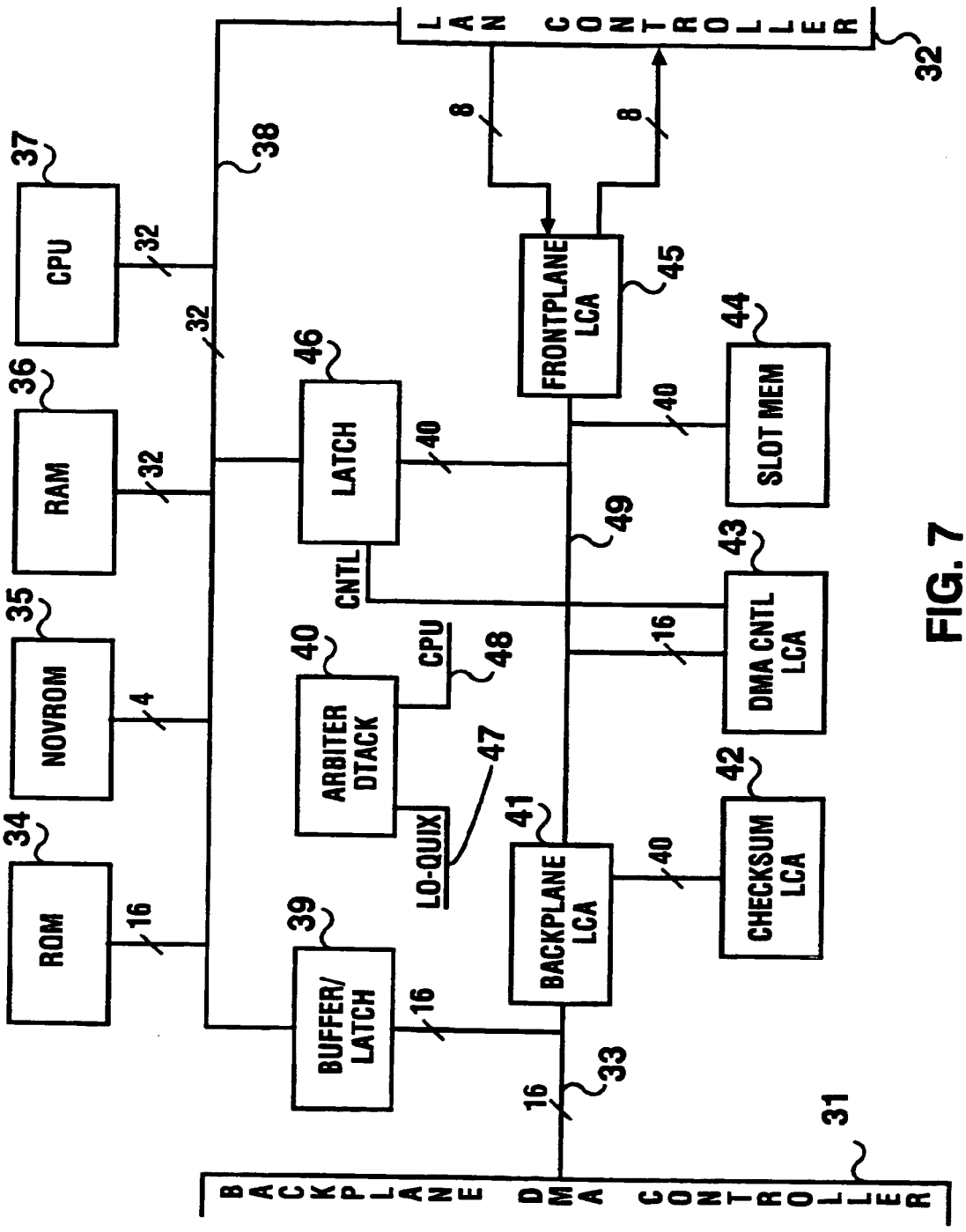

FIG. 7

BNSDOCID: <EP___0574140A1_I_>

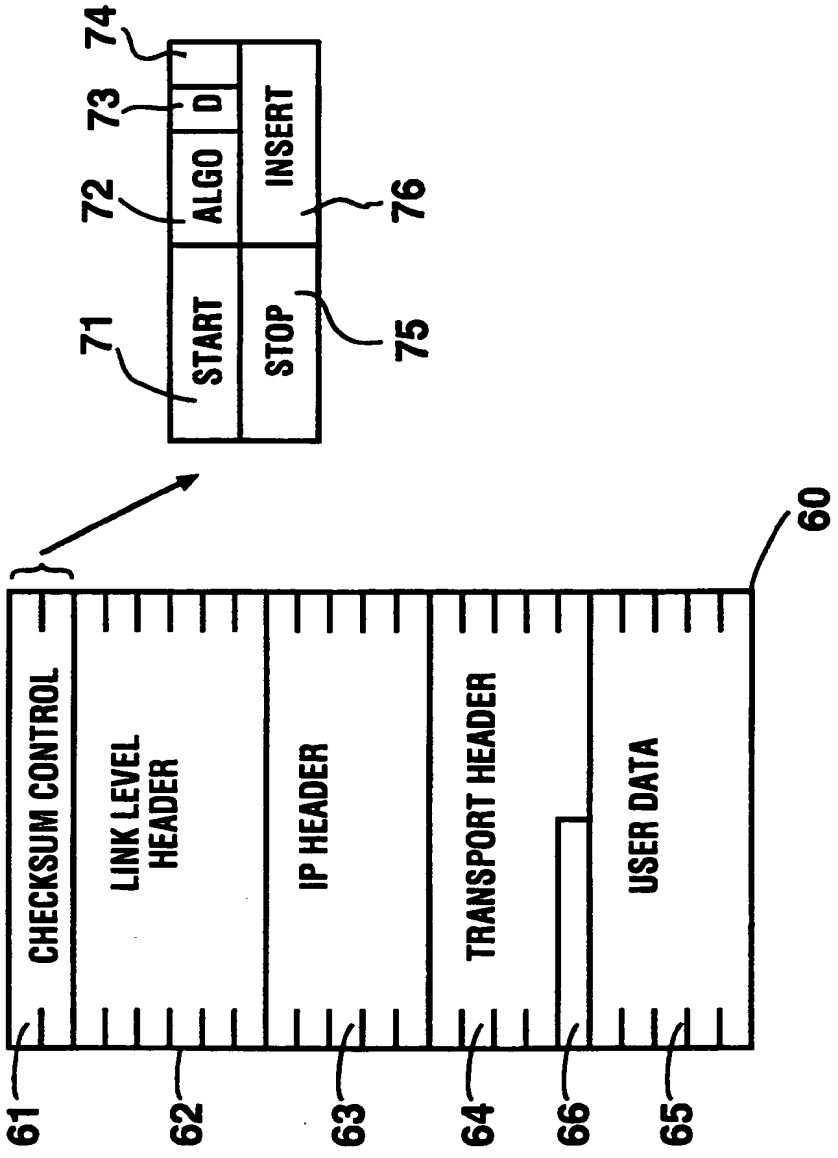

FIG. 8

BNSDOCID: <EP___0574140A1_I_>

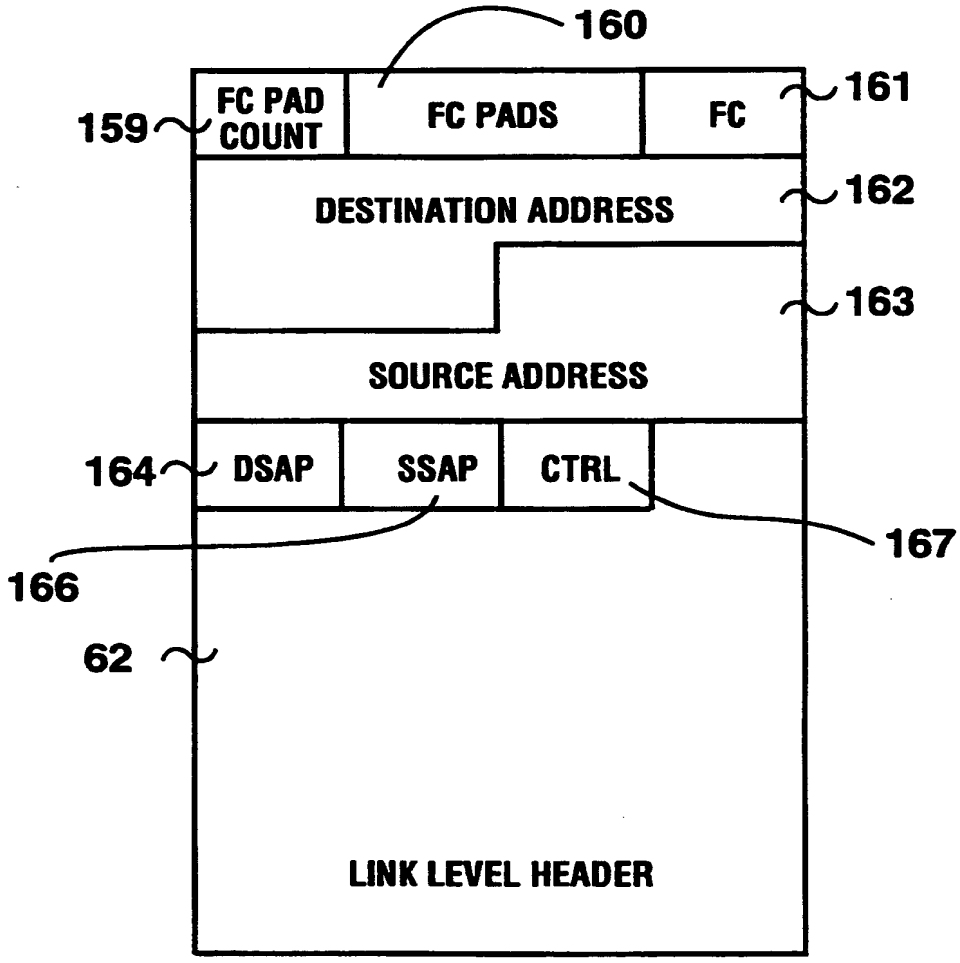

FIG. 9

BNSDOCID: <EP___0574140A1_I_>

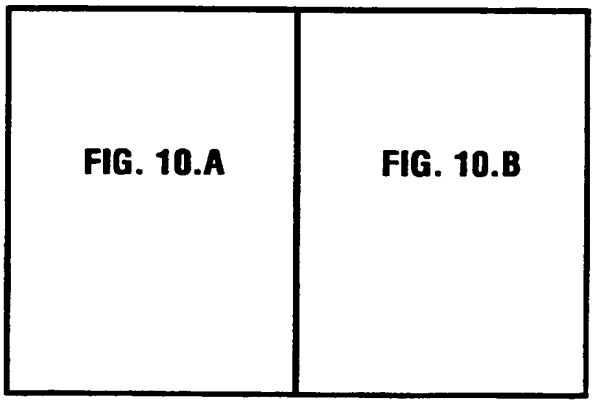

FIG. 10

BNSDOCID: <EP___0574140A1_I_>

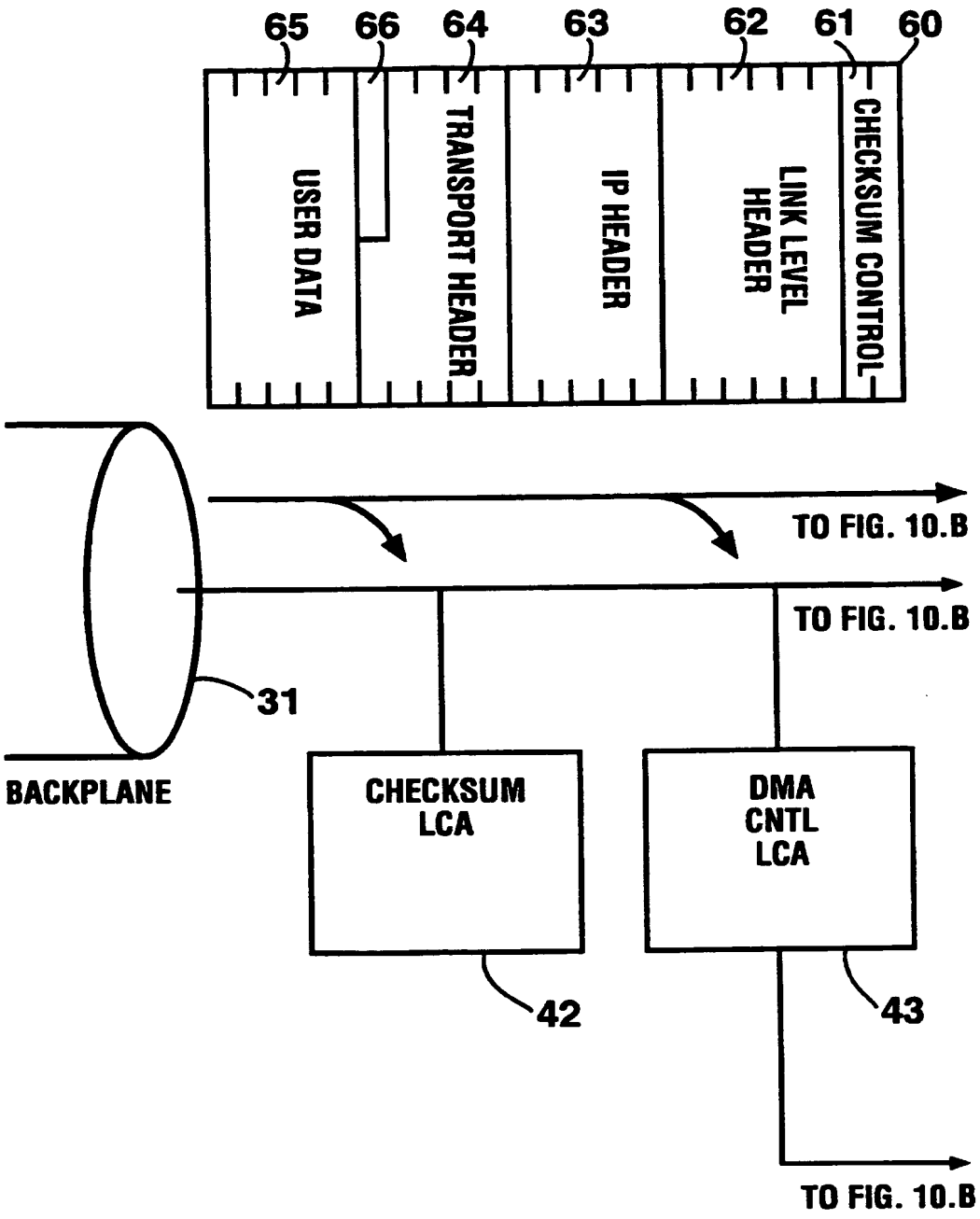

FIG. 10.A

BNSDOCID: <EP___0574140A1_I_>

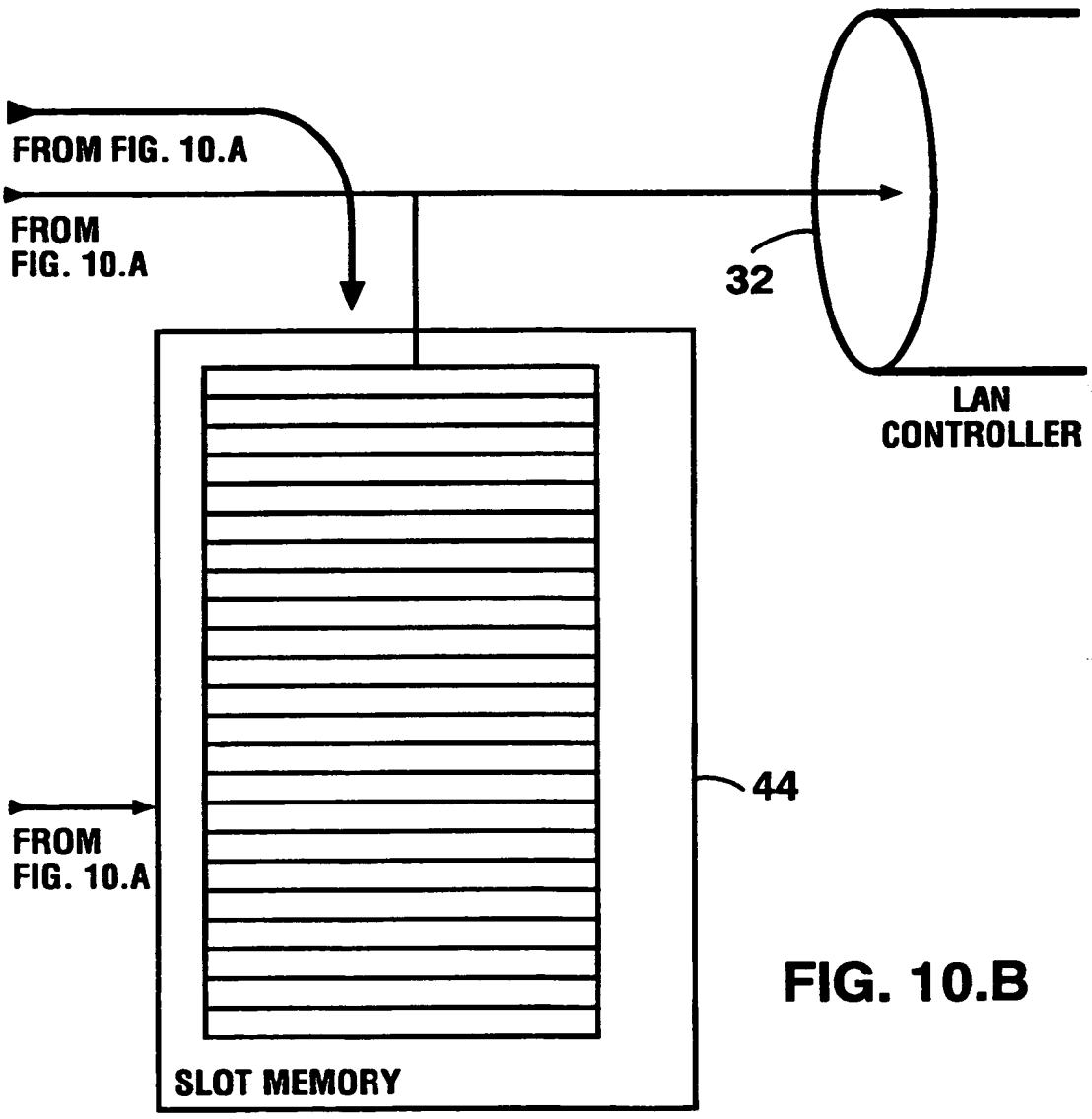

FIG. 10.B

BNSDOCID: <EP___0574140A1_I_>

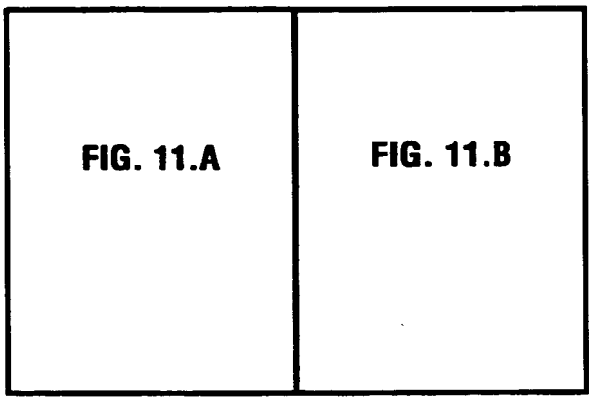

FIG. 11

BNSDOCID: <EP___0574140A1_I_>

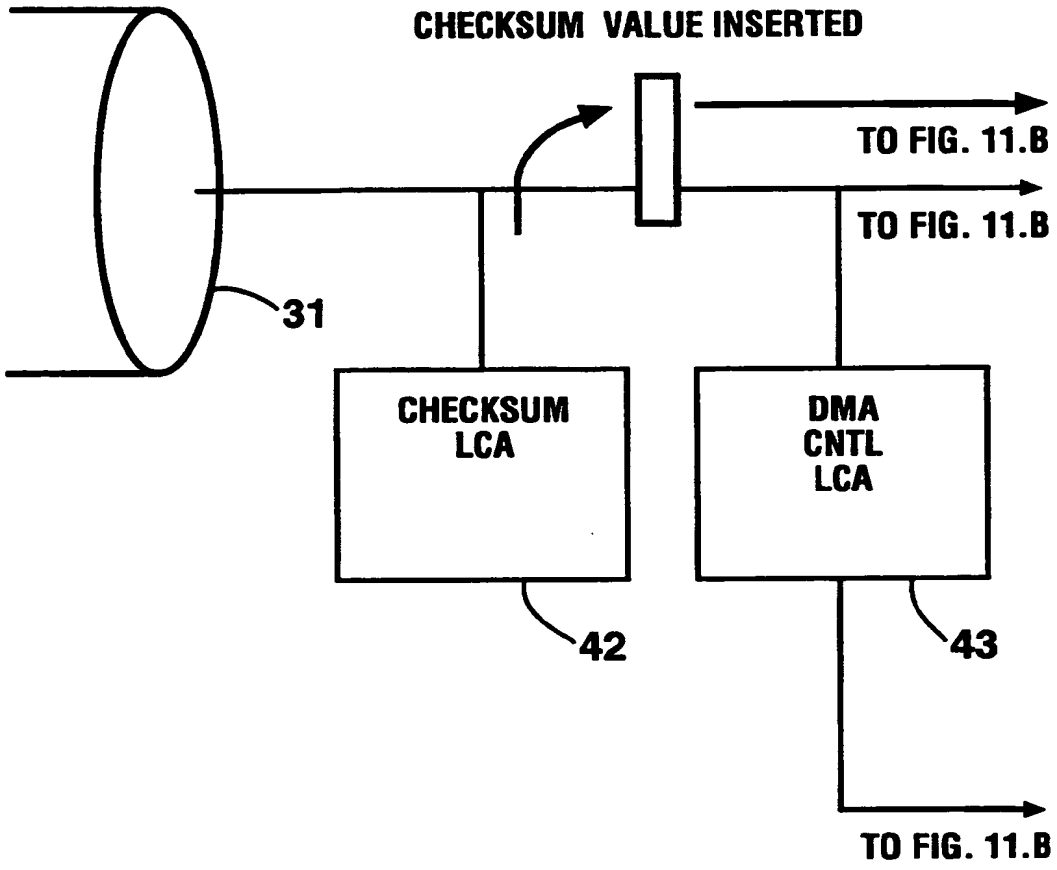

FIG. 11.A

BNSDOCID: <EP___0574140A1_I_>

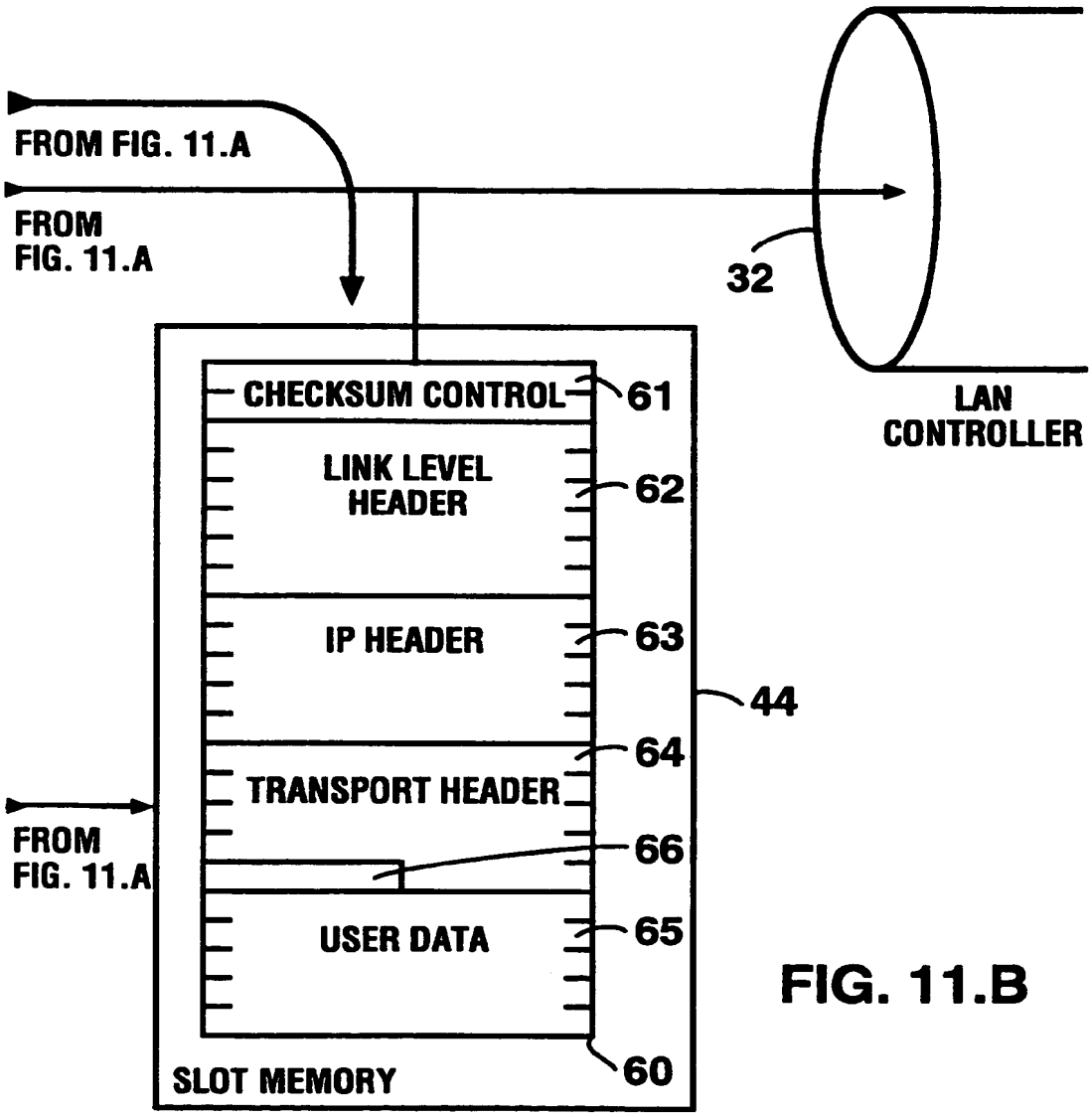

FIG. 11.B

BNSDOCID: <EP___0574140A1_I_>

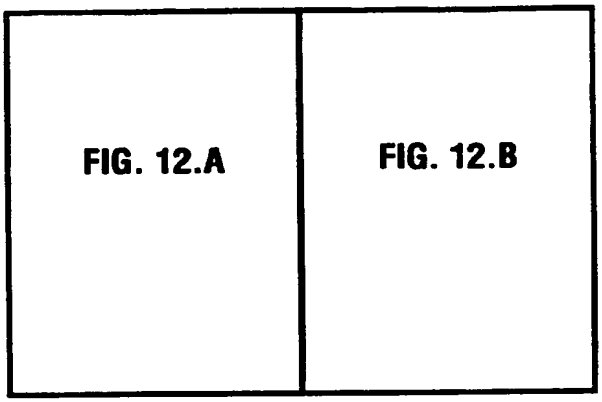

FIG. 12

BNSDOCID: <EP___0574140A1_I_>

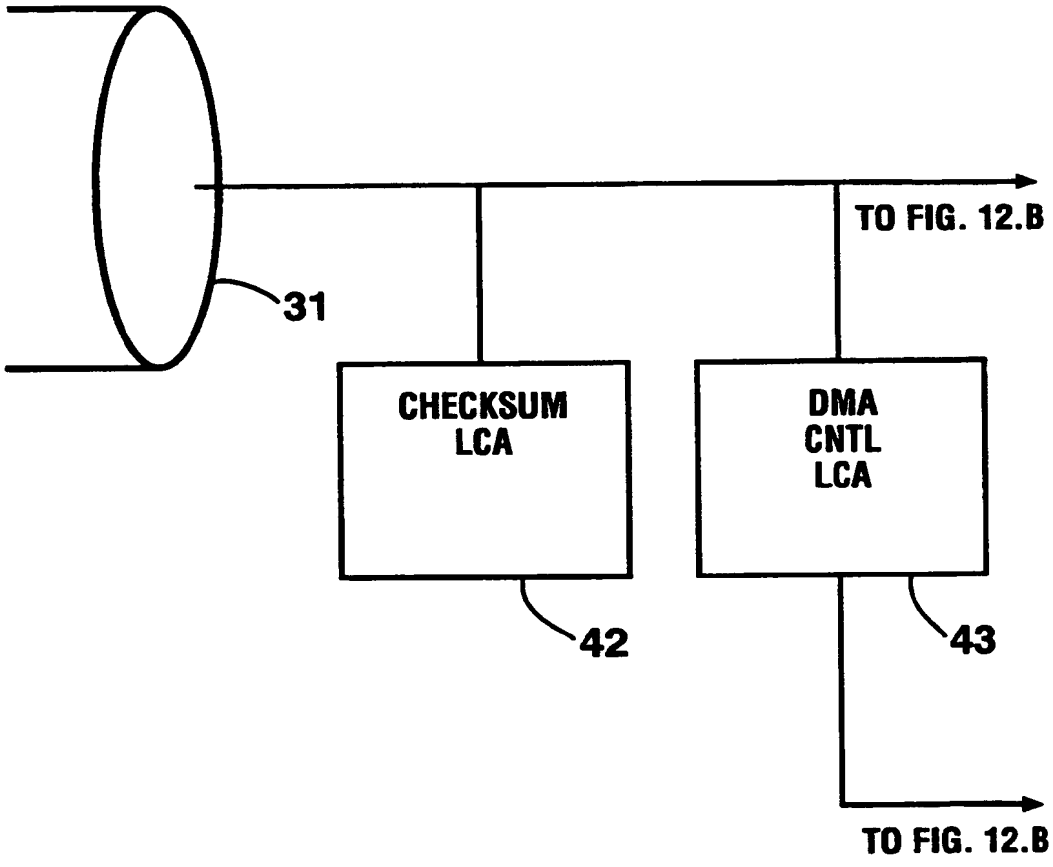

FIG. 12.A

BNSDOCID: <EP___0574140A1_I_>

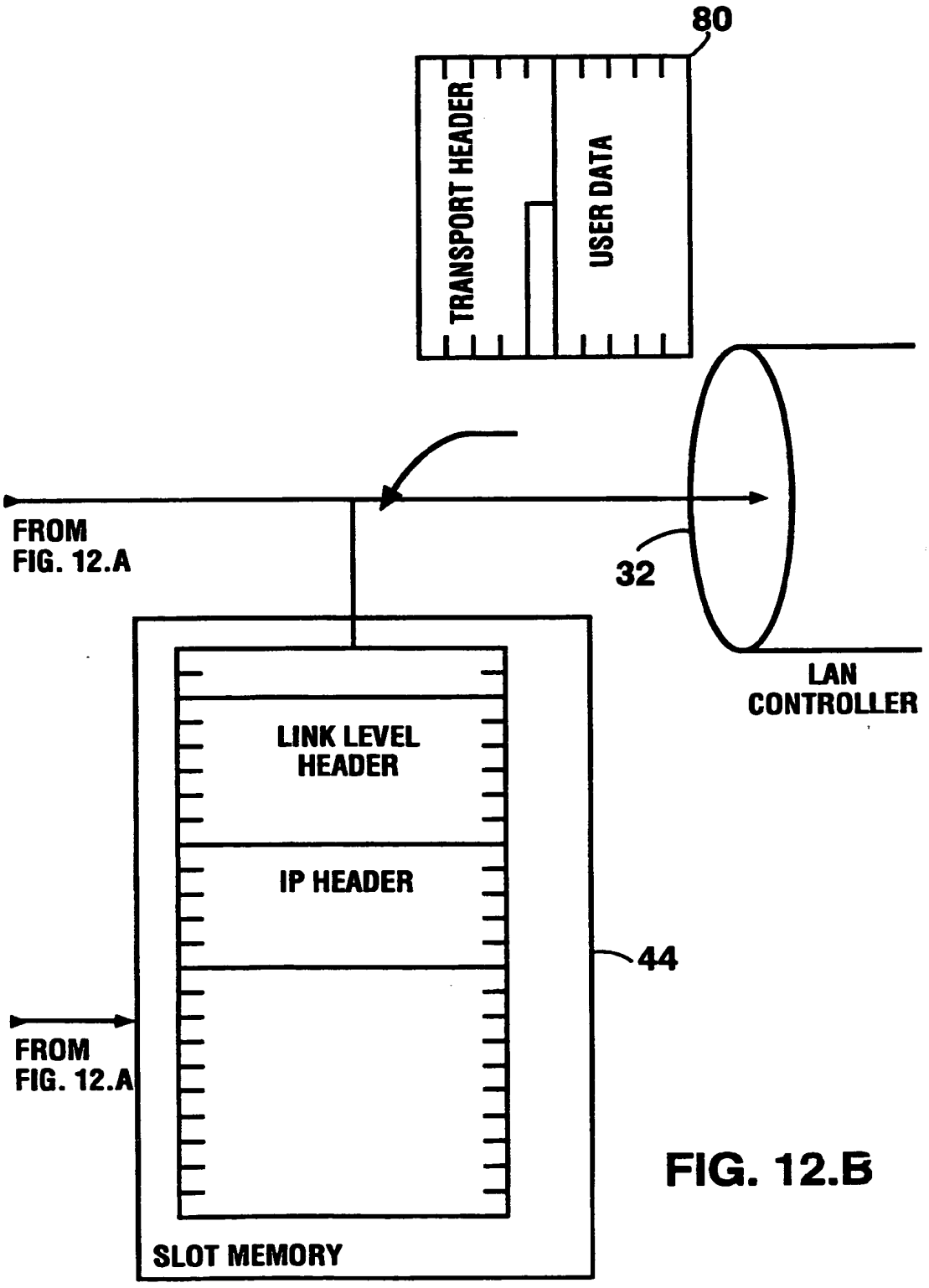

FIG. 12.B

BNSDOCID: <EP___0574140A1_I_>

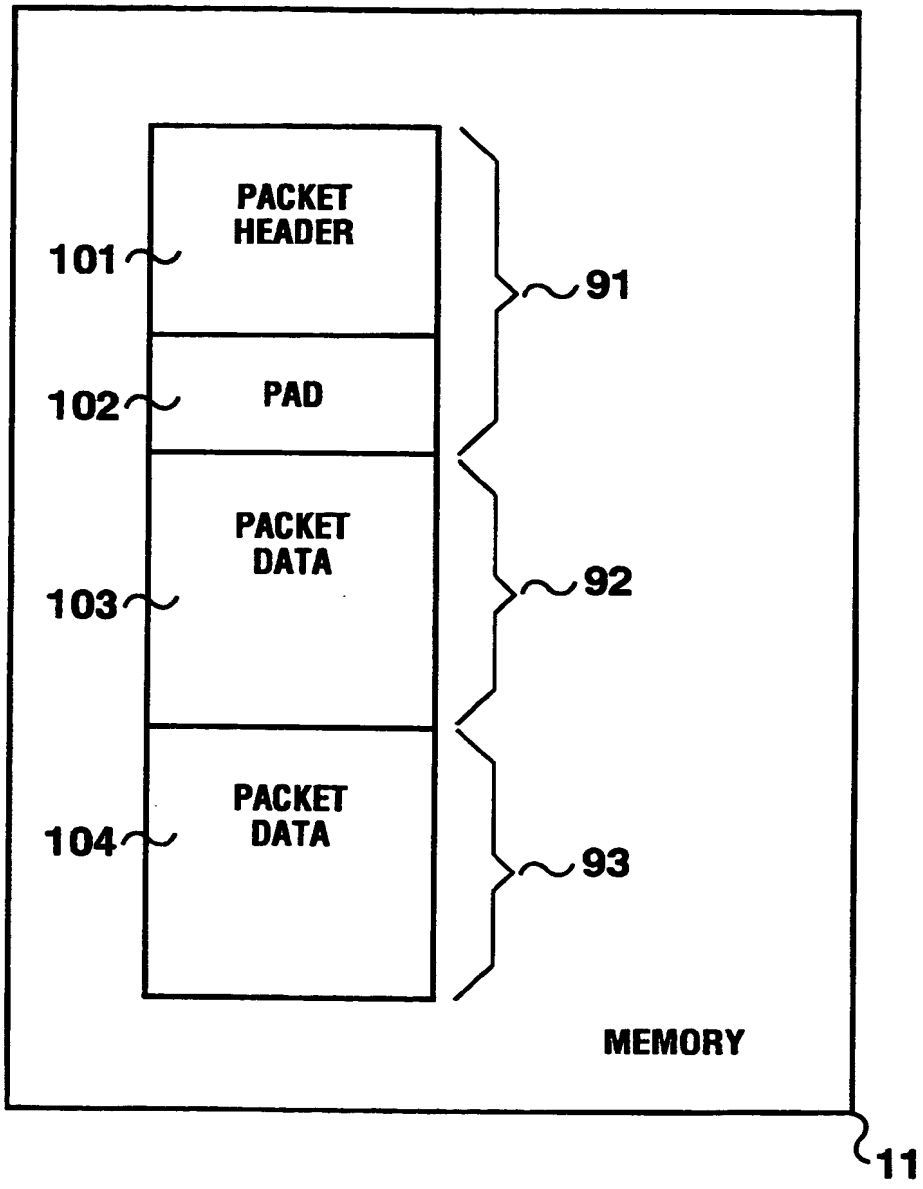

FIG. 13

BNSDOCID: <EP___0574140A1_I_>

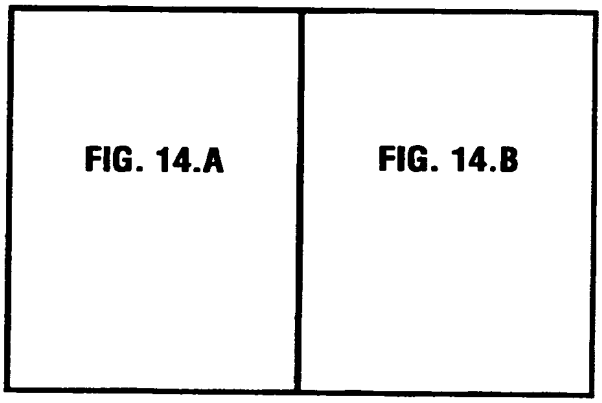

FIG. 14

BNSDOCID: <EP___0574140A1_I_>

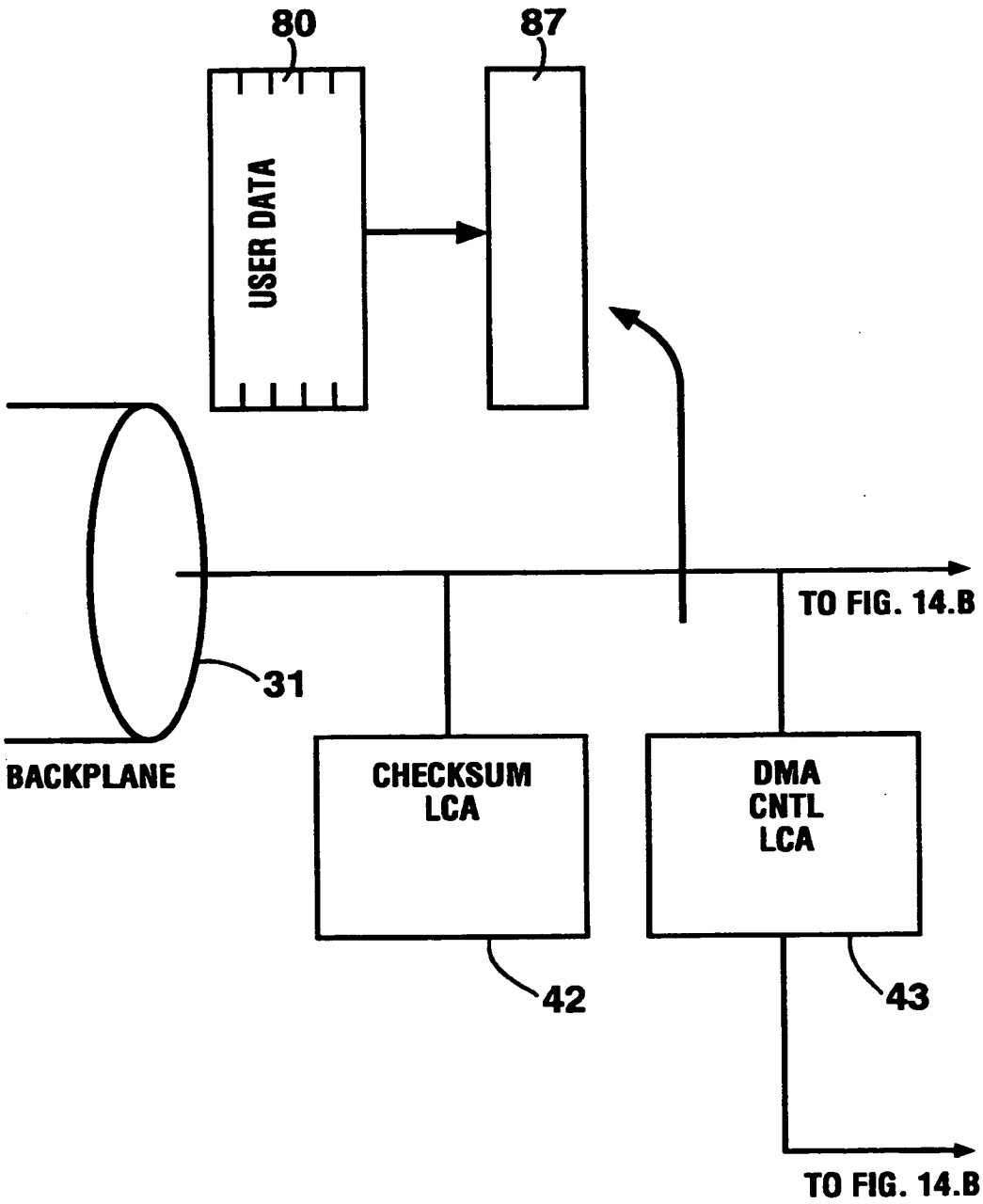

FIG. 14.A

BNSDOCID: <EP___0574140A1_I_>

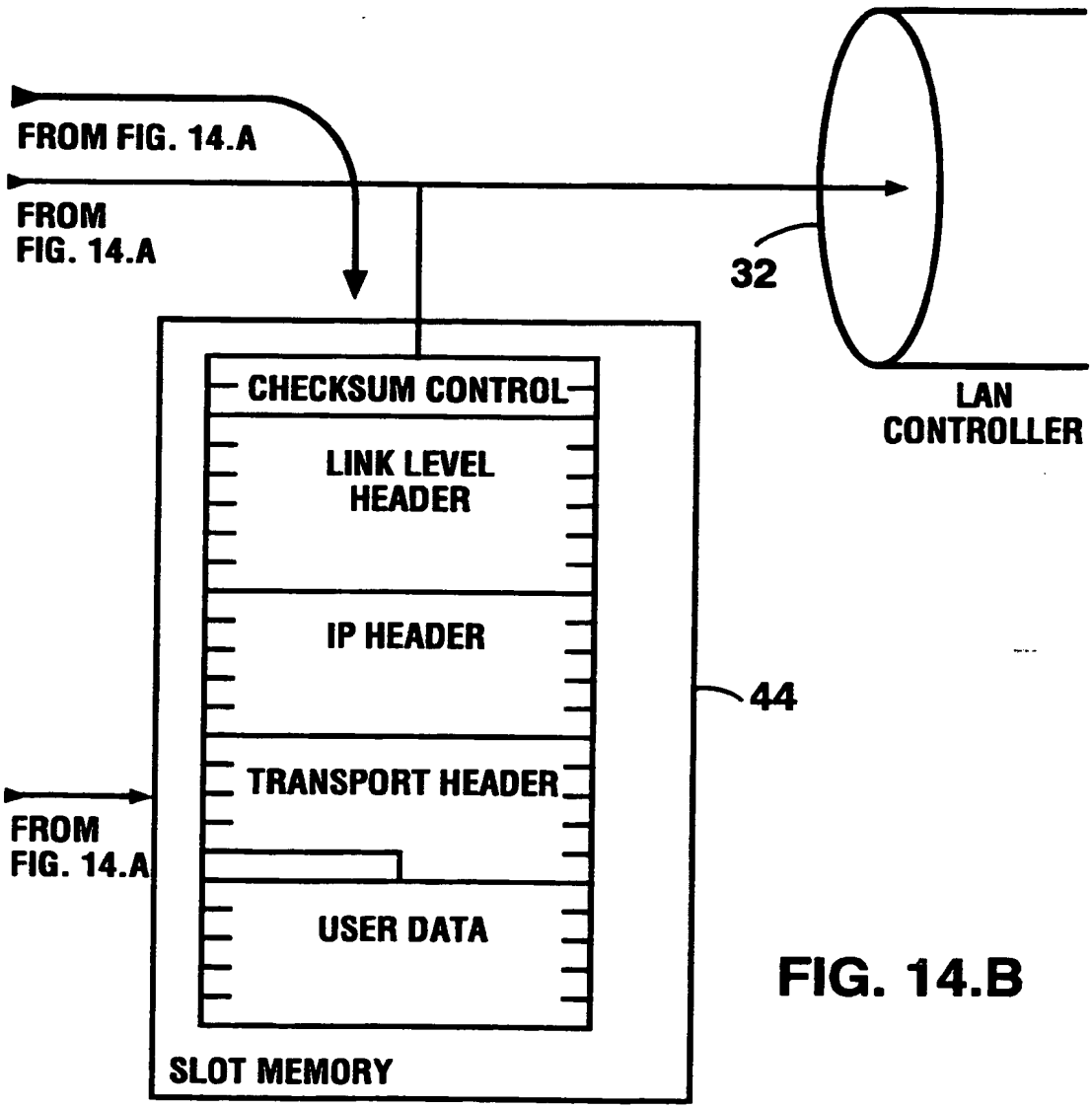

FIG. 14.B

BNSDOCID: <EP___0574140A1_I_>

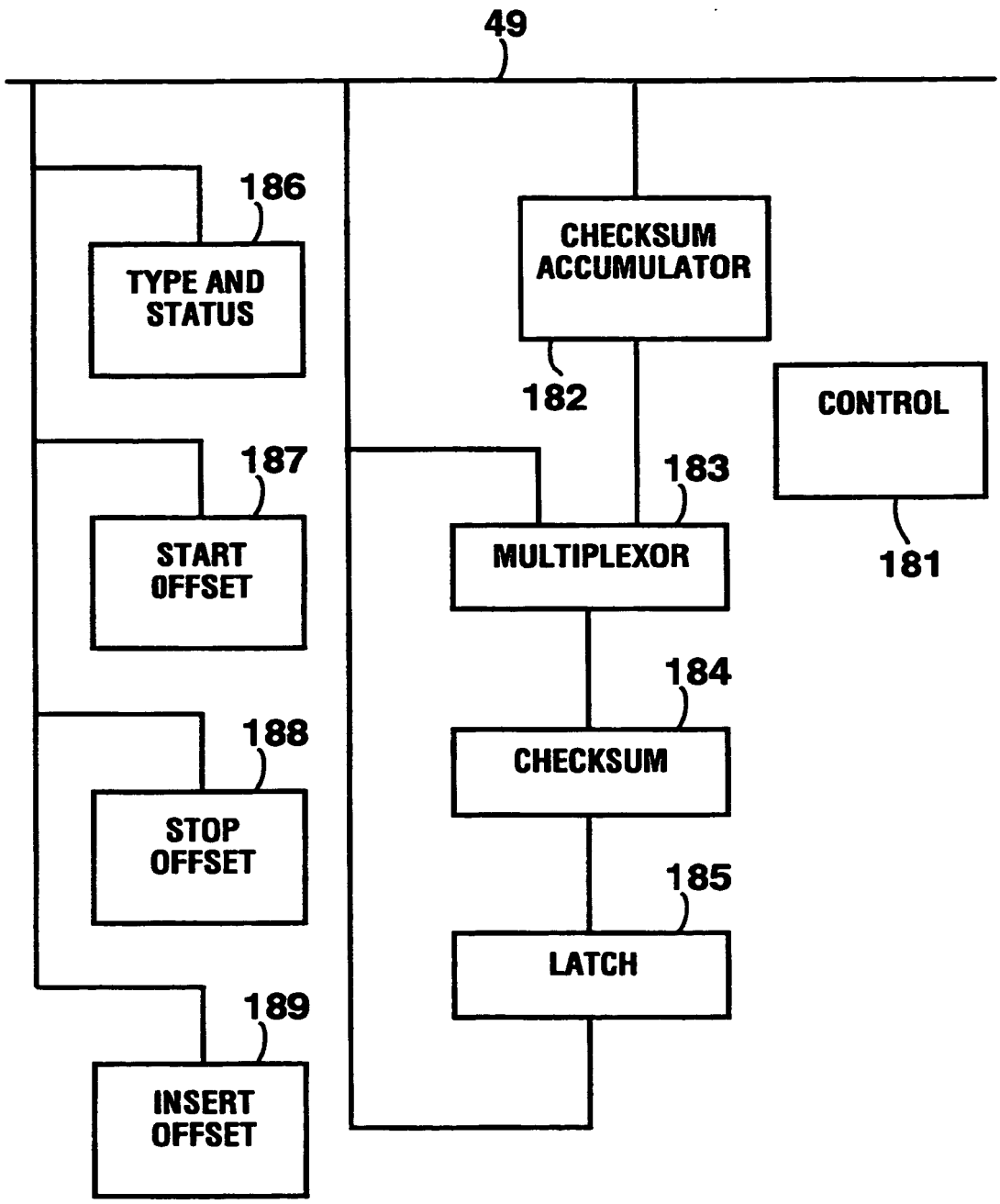

FIG. 15

BNSDOCID: <EP___0574140A1_I_>

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 93 30 3743

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.5)
X	INTERFACES IN COMPUTING vol. 3, no. 3/4, September 1985, LAUSANNE CH pages 173 - 187 R.W.DOBINSON ET AL 'INTERFACING TO ETHERNET USING VLSI PROTOCOL CHIPS' * page 179, line 11 - line 39 *	1,6,9	H04L29/06
A		3-5,7,8,10	
Y		2	

Y	T.W.PRATT 'PROGRAMMING LANGUAGES, DESIGN AND IMPLEMENTATION' 1975 , PRENTICE HALL, INC. , ENGLEWOOD CLIFFS, US CHAPTER 3 * page 54, line 1 - line 7 * * figure 3.4 *	2	
	-----		TECHNICAL FIELDS SEARCHED (Int. Cl.5)
			H04L

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	28 SEPTEMBER 1993	CANOSA ARESTE C.

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P0401)

BNSDOCID: <EP___0574140A1_I_>

THIS PAGE BLANK (USPTO)